BYU-EVE: Mixed Initiative Dialog via Structured Knowledge Graph Traversal and Conversational Scaffolding

Nancy Fulda, Tyler Etchart, William Myers, Daniel Ricks, Zachary Brown
Joseph Szendre, Ben Murdoch, Andrew Carr and David Wingate

Department of Computer Science
Brigham Young University
Provo, UT 84604

`{nfulda,tyler.etchart,william_myers,daniel_ricks,zac.brown.zcb1}@byu.edu`
`{joseph.szendre,ben.murdoch.94,andrewcarr06}@gmail.com, wingated@cs.byu.edu`

Abstract

We present BYU-EVE, an open domain dialogue architecture that combines the strengths of hand-crafted rules, deep learning, and structured knowledge graph traversal in order to create satisfying user experiences. Rather than viewing dialogue as a strict mapping between input and output texts, EVE treats conversations as a collaborative process in which two jointly coordinating agents chart a trajectory through experiential space. A key element of this architecture is the use of conversational scaffolding, a technique which uses a (small) conversational dataset to define a generalized response strategy. We also take the innovative approach of integrating the agent's self and user models directly within the knowledge graph. This allows EVE to discern topics of shared interest while simultaneously identifying areas of ambiguity or cognitive dissonance.

1 Overview

1.1 Open Domain Dialog: Challenges and Opportunities

As demand for voice technology expands, the challenges inherent in conversational AI become more pressing. Users desire voice assistants who behave less like machines and more like humans [3, 25, 29]. They don't simply want to query their devices; they seek to engage with them in complex exchanges, use them as mental sounding boards, and receive social validation in response to their statements. This type of interaction encompasses, but also goes beyond, database queries and question/answer models. In order to engage with the user fully, the system must understand the rhythms of conversational flow, maintain an internal identity and provide information that is geared to satisfy the unspoken desires of the user.

Our architecture is designed to accomplish all three of these tasks. We address conversational flow via the scaffolding technique presented in Section 5.1. Internal identity is maintained via specially tagged knowledge graph nodes as described in Section 3, and targeted information retrieval is discussed in section 7.3. Additionally, we have structured our model to sustain expansion into unrestricted dialog settings: rather than being an expert in a few specialized areas, our infrastructure is designed to apply general-purpose knowledge across a wide variety of topics and conversational settings.

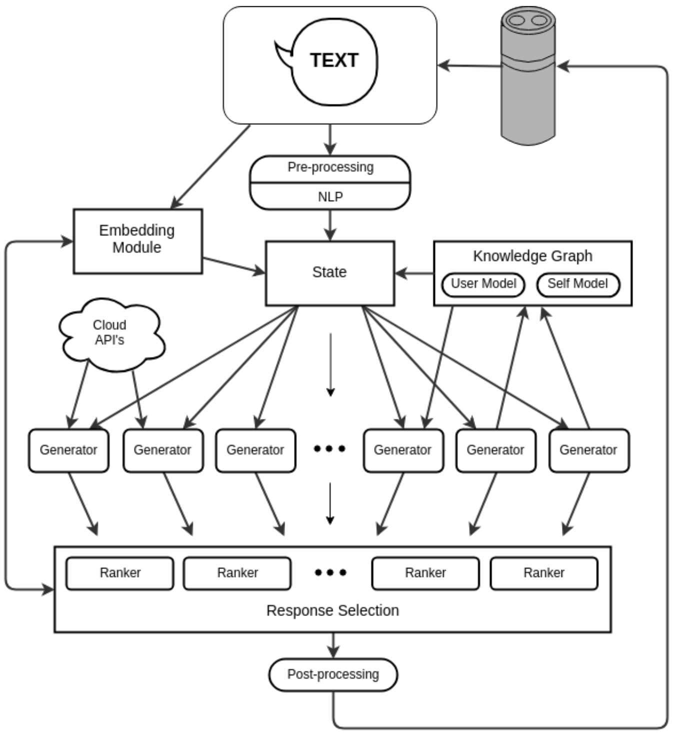

Figure 1: Our architecture is built on the premise of multiple candidate response generators vying for the attention of the dialog manager. Each generator produces a specific type of response: emotive utterance, fact retrieval, conversation offer, self-analysis, etc. The dialog manager analyzes these outputs, then selects and combines the best of them to produce a fluid, natural-sounding response.

1.2 Invisible Walls vs. Infinite Horizon: How Open Is 'Open'?

Arbitrary social dialog is a challenging problem, yet highly promising, with potential applications in areas such as education [17], political activism [13, 16], and elder care [8], among others. In order to manage the complexities inherent in unstructured, free-form dialog, many conversational systems employ an 'invisible wall' technique. The system gives the appearance of being able to discuss any topic, but is in reality guiding the conversation only to those topics that it is able to handle well. This technique has been successfully employed by [24, 31].

To make significant progress in this domain we seek to go further. Following the example set by [14], we envision an 'infinite horizon' scenario in which the system is able to navigate the full range of human conversation. This does not imply that the system must be omniscient. Rather, it means that the system is able to apply the knowledge at its disposal within a broad range of conversational contexts.

This is accomplished by housing the system's core knowledge in a single location. Instead of querying myriad APIs via dozens of specialized response generators, our architecture accumulates information within a centralized knowledge graph. Individual response generators navigate this graph in structured ways, creating a flexible, easily scalable system. In this paradigm, expanding the socialbot to handle new topic areas becomes a simple matter of augmenting the knowledge graph; no hand-coding and no new response generators are required.

We hasten to note that vision and reality are often two separate things. EVE was produced as part of Amazon's Alexa Prize Challenge, with corresponding deadlines that required her to enter production before the core structures were complete. As a result, the system is currently a hybrid between our core architectural ideas and specialized, topic-specific response generators that use direct API calls rather than relying on the central knowledge graph. These were created to satisfy consumer demand until the centralized infrastructure is finished.

1.3 Text Retrieval vs. Text Generation: The Proverbial Rock and the Hard Place

Ultimately, a dialog system exists to generate text. This text may be retrieved from a curated and/or web-scraped repository [15, 24], generated via neural network models [15, 32], or constructed using a combination of templates and placeholders [15].

Each of these methods has drawbacks. In open domain dialog, the scope of possible conversation topics makes retrieval-based systems unfeasible; there are simply too many conversational pathways. On the other hand, templates tend to sound repetitive and robotic, while generative models suffer from a lack of coherence, lack of consistency, and frequent lack of propriety.

So what's a bot to do? As an interim solution, EVE uses a combination of templates and retrieval methods, but the long-term goal is to create generative models conditioned on conversational context and knowledge graph data. In other words, we wish to control *what* the system says while using a neural network to decide *how* the idea should be expressed.

1.4 Choosing What to Say: A Question of Composition

When preparing final output for the user, we rely on the insight that conversation is a multi-channel communication stream along which not only information, but also emotive and psychological data is being passed. For example, when Alice tells Bob that she failed her math test, Bob's response is likely composite: "Oh no!" he might say, "Will you get a chance to re-take it?" The first statement acknowledges and empathizes with Alice's emotional state, while the second propels the conversation forward with a request for information.

We model this compositionality by classifying candidate responses into communication 'channels'. For each user utterance, our system attempts to generate (a) an emotive response, (b) an informative reply to the user utterance, and (c) a conversational offer such as a follow-up question or proposed conversation topic. To simplify terminology, we refer to these as emotes, answers, and offers. An ensemble of ranking algorithms scores each candidate response, and the highest-ranking emote, answer, and offer respectively are combined into a final output response.

A key element of this process is the idea of *conversational scaffolding*, the principle that a relatively small set of example conversations can be leveraged to determine which candidate responses are viable. This is accomplished by leveraging the analogical properties of embedding spaces at the conversational level, as described in Section 5.1.

2 Knowledge Representation

The storage, retrieval, and representation of knowledge is a core issue when it comes to tackling open-ended conversation. Without some form of persistent memory, it is not possible for a conversational AI to progress beyond simple social pleasantries or, at most, a generative or probabilistic model that reproduces common factual statements. On the other hand, relying on highly specialized memory retrieval mechanisms, such as an IMDb API for movies, an ESPN API for sports, or other topic-specific cloud APIs, makes it difficult to synchronize knowledge across response generators or to find relationships that span multiple topics. A unified, persistent memory allows for intelligent topic switching and strong contextualization of conversation.

While many developers are more familiar with relational database management systems (RDBMS), such as MySQL, they are not the right pick for representing general knowledge, mainly due to their rigidity. Trying to come up with tables that make sense for actors, sports, towns, books, planets, etc. is intractable; the tables would end up tremendously sparse. Graph databases, on the other hand, lend themselves to these sorts of free-form ideas. In addition, relationships between two ideas is a first-class citizen with graph databases, whereas relationships in relational databases typically require an expensive join. When representing knowledge, understanding and traversing the relationship between two ideas is crucial for open-ended conversation.

In light of these factors, we have chosen to implement persistent memory in the form of a graph database (a.k.a. knowledge graph) consisting of unique nodes connected by labeled edges. Our knowledge base currently contains 36,367,345 nodes and 164,664,878 edges reflecting a wide array of topics such as literature, biology, genetics, visual media, manufacturing, biographical information, and many others.

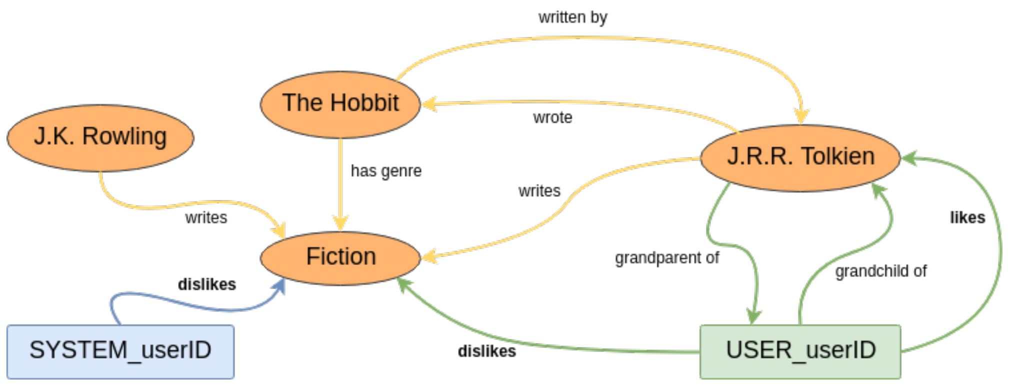

Figure 2: A representation of a portion of our knowledge graph. Orange nodes and edges correspond to general facts contained in the knowledge graph, the green node and edges represent a model of the user associated with a specific userID, and the blue node and edge represent a portion of the system model corresponding to the same userID. Bolded edge labels represent facts of interest that elicit responses from the MCI and MCE algorithms discussed in Section 3.1.

2.1 Property Graph vs. Resource Description Framework (RDF)

There are two main competing types of graph databases, property graphs and resource description frameworks (RDF).

Property Graph A property graph can best be described as a free-form web of information with a structure implied by its name. Nodes (or vertices) of the graph consist of labeled items such as "J.K. Rowling", "science fiction", or "dalmation". These nodes can have properties attached: Thus, one property for "J.K. Rowling" might be an alias property with the value "Joanne Kathleen Rowling". Some properties are important enough to deserve their own node. In these cases, the property itself is considered an edge (and will be referred to as such from here on out) and the edge's value is a pointer to another item in the graph. For example, the edge "author of" would point from the "J.K. Rowling" node to the node labeled "Harry Potter and the Sorcerer's Stone". In turn, *that* node might have an alias property with the value "Harry Potter and the Philosopher's Stone", as well as a genre edge pointing to the "fantasy" node. Storing knowledge in this way creates a flexible structure that keeps related ideas close to one another via edges.

The main benefit of using the property graph variant is the loose structure and the ability to query it using the Apache Tinkerpop Gremlin language [10]. Gremlin allows the user to make complex path queries that allow multiple degree jumps in a single query. We found this extremely useful when writing generators that wanted to tap into the knowledge graph as it allowed them to make multiple connections to a single idea, much as a human might.

RDF RDF databases are at heart, triplet stores. They come in subject-predicate-object format, such as "J.K. Rowling"-"author of"-"Harry Potter and the Sorcerer's Stone". Storing the data in triplet form make queries very quick, but limited. SPARQL [23] is the main query language for RDF databases and as eloquently put by Oracle: "The SPARQL query language is intended primarily for pattern (subgraph) matching rather than path traversal. ... SPARQL 1.1, however, still lacks the ability to reference a path directly in a query (e.g., the ability to store the result of a property path query in a path variable). Without this ability, it is not possible to match an arbitrary length path and return the path itself or perform operations based on characteristics of the path, such as path length." [6].

Even though most open knowledge graphs, like Wikidata and Conceptnet, come in RDF format and RDF queries are on average faster than property graph queries [2], we found the path traversal feature too important to pass up. As a result, we built a pipeline that converted the RDF Wikidata graph into a property graph so that we could take full advantage of the path traversal features offered by Gremlin.

2.2 Wikidata

Our core knowledge graph content is based on Wikidata [30]. Wikidata is a Wikimedia project that seeks to structure the world's knowledge. It can be thought of as Wikipedia in knowledge graph format. We originally based our knowledge graph on Conceptnet [18], but found that it was not large enough to facilitate open-ended conversation. We then turned to Wikidata, which we first had to repurpose by stripping out all non-English ideas. We then stripped out nonconversational nodes like "Wikimedia Disambiguation", and finally we added Google's Universal Sentence Embeddings [4] for the descriptions of each node. Even with all the cleaning, we found that Wikidata was somewhat esoteric in the selection facts surfaced, resulting in statements like "Do you consider cat to be a typical example of the animal painting genre?". While this problem persists today, we were able to greatly minimize it by selectively traversing only edges that make sense in the context of the current topic, i.e., when talking about movies we want to traverse "acted in" edges and not "subclass" edges.

2.3 Knowledge Graph Hosting

Our knowledge graph is managed via the Amazon Neptune service, a choice which simplifies load balancing, fault tolerance, and general updates to the graph. We use the Apache Tinkerpop Gremlin language [10] (the python variant [9]) for queries.

3 User and Self Modeling

In order to approach human-level conversation, the system must not only remember what has been said in exchanges with users, but it must *interpret* what has been said. This means the system must understand how past utterances relate to each other, to each of the conversation participants, and to potential future utterances.

We address this problem by modeling both the user and the system as unique nodes within the knowledge graph (see Figure 2). Edges attached to these nodes represent the core ideas expressed in previous utterances, such as affinities, disinclinations, and factual data including names and nicknames. As the conversation plays out, the system extracts facts from what is said by both itself and by the user and stores those facts in the knowledge graph, connecting them to appropriate nodes.

For example, if the user says "I like comics", the system connects the user's node with a 'likes' edge to a 'comics' node. Conversely, if the system generates an utterance expressing an affinity, such as "I enjoy Superman comics", the system connects its own knowledge graph node to a "Superman" node with a 'likes' edge.[1]

In this way, the system generates its own personality dynamically as a result of its interactions with the user. Because each node representing the system is user-specific, the problem of finding a personality for the system that makes a strong majority of users happy is largely mitigated, as each individual user molds their personal system model by what they choose to discuss with it.

We find this approach both novel and valuable because it places user utterances in context with past statements as well as with general world knowledge encoded in the knowledge graph. It also allows the possibility of self-reflection and conflict resolution on the part of the system. For example, if the system were to detect that the user likes "J.K. Rowling" but dislikes "Harry Potter and the Chamber of Secrets", this might prompt it to inquire more deeply as to why the user dislikes that particular book when they are generally well-disposed toward its author.

Similarly, the system might seek to offer new conversational topics based on areas of common interest. For example, if the user likes "Star Trek" and the system likes "Space Odyssey 2001", both of which are characterized as "science fiction", then the system might offer new conversation topics that fit into the same category. Alternately, if the user has expressed an affinity for "burritos" and "pizza", but a dislike for "spinach", then the system might seek to explore conversational options that are closely related to the first two nodes but relatively distant from the disliked node; see Section 7.1 for more details.

[1]One might wonder how or why the system's response generators would express an affinity that is not already present in the knowledge graph. At present, this occurs as a result of stochastic algorithms that seek to reflect the interests expressed by the user, as well as from topic-specific (i.e. non-knowledge graph) generators that express affinities while introducing new topics for discussion.

3.1 Current Implementation of Dynamic Knowledge Graph Models

The current system architecture relies on two response generators to handle user- and self-modeling functionality: DataBot and ModelQueryBot. The purpose for dividing the task of user- and self-modeling among two separate generators is primarily due to time constraints on how quickly the generators must return a response to the user and the latency (usually about 50-100ms database + 50-300ms network) involved in querying and updating the Neptune Knowledge Graph API.

Populating Models - DataBot DataBot is designed to extract as many knowledge graph facts about the user and the system as possible from the conversation history and to use those facts to populate the user and self models. As of the writing of this paper, DataBot uses a combination of named entity recognition and regular expressions to search for three specific edge types: 'likes', 'dislikes', and 'called'. 'Called' edges represent the name of the user or the system. 'Likes' and 'dislikes' edges represent the stated preferences of the user and the system.

Once a potential edge is found, Databot examines the utterance to identify its associated nodes and pushes a new fact of the form (node1, edge, node2) - which we will refer to as a 'triple' - to the appropriate model within the knowledge graph. The new triple is immediately visible to all other response generators within the system, which can use the stored information as they please.

Employing Models - ModelQueryBot ModelQueryBot's design is twofold: (1) it ensures that no contradictory information exists in the system's user and self models, and (2) it generates novel responses using those models. To accomplish the first objective, ModelQueryBot employs an algorithm which we call Model Error Detection (MED) to identify contradictory information in the user and self models based on predetermined rules. One such rule is that the user model should never have a 'likes' edge and a 'dislikes' edge connecting the same two nodes in the knowledge graph. If any contradictory information is identified, ModelQueryBot deletes the contradictory information and prompts the user for clarification in a generated response.

If no contradictory information has been found, ModelQueryBot will generate a response based off of the user and self models using algorithms we introduce as Model Connection Identification (MCI) and Model Content Extension (MCE). Using MCI ad MCE, ModelQueryBot first examines the primary nodes and edges directly connected to the user and user-specific system nodes, as well as all of the secondary nodes and edges directly connected to the primary nodes. ModelQueryBot then uses one of the identified triples to populate one of several relevant, predetermined templates; for example, one such template reads as "Oh, you don't like (node1) and since (node1 edge node2) do you also not like (node2)? I'm rather fond of (node2)". As the user and self models grow in complexity, so does the level of reasoning apparent in the responses output by ModelQueryBot.

Algorithm 1 - Model Connection Identification (MCI)

```
Inputs:
  $K = (N, E, L, T)$
      $N \subset \mathbb{R}$
      $E \subset \mathbb{R}$
      $L = range(f)$ where $f : (N \cup E) \to L \subset S$        ▷ Labels, mapping Nodes $N$ and Edges $E$ into natural language $S$
      $T = \{t | t = (n_i, e_j, n_k); n_i, n_k \in N; e_j \in E\}$
  $M_1 \subset K, M_1 = (N_1 \subset N, E_1 \subset E, L_1 \subset L, T_1 \subset T)$
  $M_2 \subset K, M_2 = (N_2 \subset N, E_2 \subset E, L_2 \subset L, T_2 \subset T)$
  $n'_1 \in (N \cap M_1)$
  $n'_2 \in (N \cap M_2)$
  $X$ = A set of pre-determined epsilon pairs of interest
  $R$ = A set of 'referenced' pairs of triples
Parameters:
  $\beta \in \mathbb{Z}$        ▷ The maximum number of triples in returned sets of triples to be resolved
Output:
  $D$

1: $A \leftarrow \{a = \{t_1, t_2\} : t_1, t_2 \in T'$ satisfy $n'_1 \in t_1, n'_2 \in t_2$, and $(e_u, e_v) \in X$ for any $e_u, e_v \in E$ s.t. $e_u \in t_1$ and $e_v \in t_2\}$
2: $A' \leftarrow A - (A \cap R)$
3: while $b < \beta$ do
4:     $D_b \leftarrow \{d_b = \{t_0, t_1, ..., t_{b-1}, t_b\} : t_{b-1} = (n_i, e_j, n_k) \in d_{b-1} \in D_{b-1}, t_b = (n_k, e_y, n_z), i \neq z, D_0 = A'\}$
5:     $D'_b \leftarrow \{d'_b = \{t_0, t_1, ..., t_{b-1}, t_b\} : \{t_0, t_b\} \in A, d_b \in D_b\}$
6:     $D \leftarrow (D \cup D'_b)$
7: end while
8: return $D$
```

Model Connection Identification (MCI) MCI is an algorithm for finding all of the connections between two nodes in a knowledge graph within a given search radius. MCI takes the general knowledge graph $K = (N, E, L, T)$ as input, where N is a set of indices corresponding to nodes, E is a set of indices corresponding to different edge labels, L is the set of labels that correspond to the indices in N and E, and T is the set of all fact triples of the form (node1, edge, node2) in the knowledge graph. MCI also requires two sets of triples that model entities (either distinct or identical) which we refer to as M_1 and M_2 as input, as well as the nodes n'_1 and n'_2 that represent the entities themselves. Finally, a set of pre-determined epsilon pairs of interest X (in ModelQueryBot, it is often the case that $X = \{(likes, likes), (likes, dislikes), (dislikes, likes), (dislikes, dislikes)\}$), and a set R of pairs of knowledge graph triples that have previously been addressed with the user are input into MCI. Using its inputs as well as a parameter β to control the depth of its search, MCI collects a set D of sets of triples from the knowledge graph that connect n'_1 and n'_2 together which is then returned as output. $\beta = 2$ is the maximum search window used in ModelQuerybot currently. We define MCI more formally in Algorithm 1.

Model Content Extension (MCE) MCE is a special use-case of MCI; we write MCE as its own algorithm here to highlight a few of the various uses for MCI that go beyond merely finding relevant factoids relating two specific nodes which can then be used to generate text. In MCE, MCI is given K, X, and R as normal (currently with the parameter $\beta = 2$ often kept constant as well), but then receives only one model M of a particular entity and the entity's associated node n'. By performing MCI on only one entity, instead of trying to connect two arbitrary entities as in MCI, MCE effectively tries to find logical connections within the nodes of M; in essence, we currently use MCE to try and find causal relationships between the "likes" and "dislikes" of the user (and potentially to give causal relationships for the "likes" and "dislikes" of our own system in future work). Therefore, ModelQueryBot uses MCE to detect particularly interesting subtleties in its model of the user - for instance, MCE would be used to ask why a user would tell us that they both "like ice cream" and "dislike dairy products" at the same time, or to ask the user if the reason they "like J.R.R Tolkein" is because they also "like the Hobbit". MCE is expressed formally in Algorithm 2.

Algorithm 2 - Model Content Extension (MCE)

```
Inputs:
    $K$                                  ▷ The Knowledge Graph; see the description of $K$ in Algorithm 1 - MCI for more details
    $M \subset K$
    $n' \in (N \cap M)$
    $X$
    $R$ = A set of 'resolved' pairs of triples
Parameters:
    $\beta \in \mathbb{Z}$
Output:
    $D$

1: return $D \leftarrow MCI(K = K, M_1 = M, M_2 = M, n'_1 = n', n'_2 = n', X = X, R = R, \beta = \beta)$
```

Model Error Detection (MED) MED is the algorithm we use to remove contradictory facts from our user and self models. Similarly to MCE, MED takes as input the Knowledge Graph K, a single entity model M and its associated entity node n', as well as X, which in this case represents a set of pre-determined contradictory epsilon pairs that should never be found connecting the same two nodes simultaneously (currently, MED is given $X = \{(likes, dislikes), (dislikes, likes)\}$ in ModelQueryBot). Furthermore, MED takes in the set X' which represents a set of pre-determined 'singular' epsilons, which should connect a given node (in the case of MED, this node is n') to only one other unique node; in ModelQueryBot, MED receives $X' = \{called\}$. Given its inputs, MED returns a set C representing all of the 'contradictory' knowledge graph triples in the input Model. MED is expressed formally in Algorithm 3.

3.2 A Note on Privacy

When users interact with our system, they implicitly agree to share personal information (e.g., opinions, facts about themselves, etc.) in the form of natural conversation. Our system attempts to harvest this information in a way that will mold and personalize the experience for each user by adding it to the general knowledge graph. However, if at any point this becomes an issue or a

Algorithm 3 - Model Error Detection (MED)

```
Inputs:
    $K$                    ▷ The Knowledge Graph; see the description of $K$ in Algorithm 1 - MCI for more details
    $M \subset K$
    $n' \in (N \cap M)$
    $X$
    $X'$
Output:
    $C$

1: $C_{pair} \leftarrow MCD(K = K, M = M, n' = n', X = X, R = \emptyset, \beta = 1)$
2: $C_{singular} \leftarrow \{p = \{t_1, t_2\} : t_1 \cap t_2 = \{n', e_j\}$ with $e_j \in X'$, and $t_1, t_2 \in T\}$
3: $C \leftarrow (C_{pair} \cup C_{singular})$
4: return $C$
```

user wishes to have their information deleted, it is trivial to delete the system node and user node associated with their ID, which will in turn delete all edges associated with them and effectively remove any personal information collected.

4 Response Generation

EVE's response generators fall into three broad categories: *Emotive generators*, which seek to identify and empathize with the user's mood, *Knowledge Graph generators*, which seek to find connections between user utterances and related nodes within the knowledge graph, and *API* generators, which query local repositories or online data repositories in order to respond to user inquiries. We also maintain a small set of *Fallback generators* designed to engage when the system fails to find an appropriate response in one of the other categories and a set of *Scripted generators* that engage when predefined text or sensitive topic material is detected.

Generators are able to pass meta-data back and forth to one another between responses. For example, if a response generator makes use of a keyword uttered by the user, it can pass the keyword in the meta-data so other generators know what the topic of conversation was. Response generators can also request priority from the dialog manager and preserve internal state between dialog turns.

Long-term, it is expected that most API generators will be replaced with knowledge graph generators; possible exceptions being news and social media-based generators. At that point, API queries would be used primarily to populate the knowledge graph with new data.

4.1 Emotive Generators

Our emotive generators seek emotional cues in the text and attempt to respond appropriately. For example, if the user says "I like fluffy bunnies", the SmartEmote generator might respond "No kidding? Me too!" Alternately, if the user makes a statement like "I feel sad", the EmoBot generator will engage this emotion with a statement like "I'm sorry. You deserve to feel better than that."

4.2 Knowledge Graph Generators

The knowledge graph generators are the core of our infrastructure, and are still in active development. DataBot and ModelqueryBot are described in Section 3.1. The remaining knowledge graph generators are as follows.

FirstDegreeFact
Takes a node of interest and finds all relevant (by topic) first degree connections. Randomly samples a connection and then uses a template to talk about the fact.

FirstDegreeConversationStarter
Similar to the FirstDegreeFact except that it takes the first degree connection and tries to transition the conversation in a relevant direction.

SecondDegreeFact
Works just like the FirstDegreeFact, with the main difference of using both the first degree and second degree connections instead of just first degree connections.

The strength of the knowledge graph generators is perhaps best illustrated by EVE's handling of the 'Literature' topic. Our infrastructure does not use any book-based APIs or knowledge repositories, and yet, using only the knowledge graph generators and a small set of hand-coded conversation starters, the system is able to respond to book-related utterances so well that 'Literature' is one of our highest-rated conversation topics.

4.3 API Generators

As discussed in Section 1.2, the API generators were implemented primarily as placeholders to satisfy customer demand until the knowledge graph infrastructure is fully on-line. They use an elementary combination of conversation templates combined with API calls to OMDb, TMDb, ESPN, Washington Post, DuckDuckGo, Wikipedia, and Amazon Evi. Many of the API generators maintain an internal state regarding the current conversation topic, which questions were recently asked, etc.

5 Response Evaluation

Free-form conversation is variational, not deterministic; this means that there is no single correct response to a given utterance or sequence of utterances (although there are certainly plenty of incorrect ones). Rather, the set of acceptable and/or excellent responses can be modeled as a probability distribution across the range of possible utterances.

For example, the question "Have you seen any good movies lately?" can be appropriately answered by statements including "Yes", "No", "I saw Iron Man last night", and "Are you crazy man? You know I'm afraid of theaters." Each of these responses is equally valid, although not equally likely. In contrast, responses such as "copper filings", "The United States was founded in 1776", or "My flashlight batteries are empty" are so unlikely as to be implausible.

Within the context of conversational AI, this is both good and bad news. It's good because our response selection criteria need not concern itself with finding the *most correct* response (because many possibilities are equally correct); It is sufficient to identify the subset of possible responses that are relatively likely to occur. From these, a final response can be selected based on criteria such as length, uniqueness, and literary quality.

Unfortunately, the variational nature of conversation makes it difficult for a naive neural network, or even a well-structured variational neural network, to learn a pattern of correct conversational responses. We address this problem by introducing *conversational scaffolding*, a technique that allows a small set of sample conversations to guide the bot's overall behavior.

5.1 Conversational Scaffolding: An Analogical Approach to Response Prioritization

Most practitioners of natural language understanding are familiar with the analogical coherence demonstrated by word2vec [19], GLovE [22] and other word-level embedding spaces. The basic principle is simple: (1) Take a word. (2) Convert it into a vector using a neural network trained based on word co-occurrence patterns. (3) Now examine its relationship to other words in the embedding space. Mikolov et al. have shown and other researchers have confirmed that within these embedding spaces, simple mathematical operations are sufficient to solve analogical queries [20, 27, 11, 12].

Our conversational scaffolding algorithm is based on the observation that similar analogical coherence can be found in sentence-level embedding spaces, albeit not as precisely or compactly structured. It is not, generally speaking, possible to mathematically compute "I like apples" - "because they are red" + "I like pears" and end up at the embedding for "because they are green"; but when one does the math, one ends up in the *general vicinity* of pears and green-ness.

Our key insight is the idea that within the context of response prioritization, *general vicinity*[2] is sufficient to identify candidate responses that are viable and conversationally coherent with respect to each user utterance. The basic principle is simple: A pair utterances representing the two most recent dialog turns is converted into vector representations using Google's Universal Sentence Encoder [4]. These vectors are then matched against a conversational dataset in order to identify likely subsequent

[2]Here, *general vicinity* means 'within a threshold distance'. We used a euclidean distance metric with $.6 * ||\mathbf{h}||$ as the threshold, where $||\mathbf{h}||$ is the number of utterences in the conversation history window.

sentences. (In theory, this task could also be attempted using a sequence-to-sequence model. However, the scarcity of high-quality training data combined with the variational nature of conversation makes this a challenging proposition. Hence our decision to use a scaffolding algorithm.)

Our conversational scaffolding methods require a high-quality conversational dataset that exemplifies the type of discourse the bot should emulate. As it turns out, these are hard to come by. See the next subsection for further details.

5.2 The Chit-Chat Dataset

In order to find a suitable dataset for our scaffolding technique, we first examined the Cornell Movie-Dialogs Corpus [5] and related datasets [7, 21]. Unfortunately, movie scripts proved unsuitable for our purposes because of two key drawbacks. (1) The text and subject matter tend to be overly dramatic rather than reflecting normal day-to-day dialog, and (2) The characters often speak in response to physical events within the scene; thus the dialog when taken out of context becomes meaningless. Social media dialogues like Twitter [28] or online forums like Reddit [26] avoid the key pitfalls of movie scripts, but also tend to contain highly controversial material, personal insults, and incendiary opinions: not exactly the behaviors we wish our socialbot to emulate.

After reviewing and eventually rejecting the available turn-based datasets, we decided to construct our own. We outsourced our data collection methods to students at our university in the form of a competition [34] in which students were randomly paired with each other in an online chat forum and asked to discuss one of several pre-defined topics. To heighten interest and engagement we offered various prizes, and these prizes were awarded to users with the highest number of quality interactions (quality being defined as a complex relationship between post length, word length, and other factors).

The resulting Chit-Chat dataset contains over 90,000 utterances from almost 1200 different users. The data is almost entirely free of offensive or derogatory statements, however it is a little noisy; we set out to capture authentic examples of human conversation, but the task itself became a topic of conversation. Chatters tended to discuss the competition's prizes, the scoreboard, and their desire to win, which subverts the actual goal of the task and creates the possibility of strange dialogue patterns in any system that uses it. (e.g. imagine a socialbot talking about a competition to provide data for a socialbot). Overall though, the data generated from our simple competition is quite good, and this dataset has become a solid training basis for multiple components in the system architecture.

Our conversational dataset is unique in that, unlike Reddit and other popular conversational datasets, there is a minimum of personal attacks and mean-spirited discussion. During account creation, each user committed to maintain high standards of behavior and agreed to let us use and share their conversation data for research purposes. We will publicly release the dataset at a later date, pending further stages of our Chit-Chat competition. (For example, we may want to increase the size of the dataset and/or filter out references to the contest, scoreboard, and prizes).

5.3 Scaffolding Algorithms

We are actively exploring three algorithms for response prioritization. All three methods rely on the Chit-Chat dataset to provide conversational examples: however, the method for selecting a follow-on sentence varies. We compare the performance of all three algorithms to a multi-layer network trained to predict the correct subsequent sentence embedding for a conversation history of 2.

For simplicity, the algorithm descriptions below assume a conversation history of 1, meaning the algorithm is provided only the user's most recent utterance. However, we have found the latter three algorithms more effective when given a conversation history of 2, consisting of the user's utterance *and* the most recent utterance provided by the socialbot. This allows the algorithm to contextualize and respond meaningfully to information-sparse utterances like "yes" or "I'm not sure".

All three algorithms assume the availability of a set of candidate responses g_i, which represent the outputs of the socialbot's various response generators for user utterance c.

Naive Offset This scaffolding algorithm is based on the simplifying assumption that the closest match for the user's utterance within the Chit-Chat dataset is always paired with an optimal response. (In reality, Chit-Chat utterances with a slightly larger distance from the user utterance are often paired with superior responses; this is addressed in the scattershot and flow vector methods, below.)

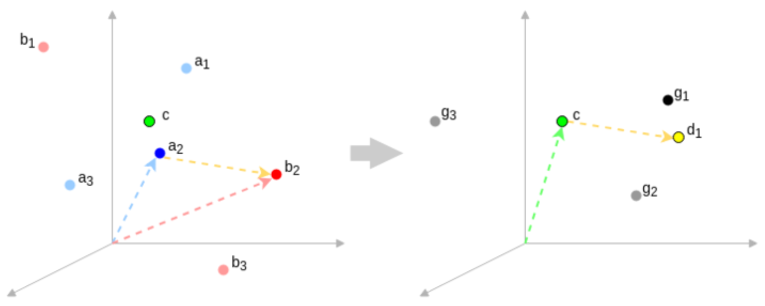

Figure 3: Naive Offset, where: c (green) represents the embedded utterance, a_i (blue) represent the nearest embedded utterances from the Chit-Chat Dataset, b_i (red) represent the associated embedded response to a_i in the Chit-Chat Dataset, d_1 (yellow) represents the 'ideal' response, and g_i (grey and black) represent embedded responses generated by our system with g_1 (black) representing the response selected by the Naive ranking strategy.

Algorithm 4 Naive offset

```
Inputs:
$\vec{c}$ = Embedded user utterence
$\vec{r}$ = Candidate responses produced by the generators.
$C$ = Chit-Chat dataset

Output:
$S = \{s_1 \dots s_i \mid s \in [0,1]\}$ where $s_i$ is the score for $r_i$

1: $\vec{a} \leftarrow min(dist(\vec{u}, C))$            ▷ Where $dist$ is any valid distance metric.
2: $\vec{b} \leftarrow$ Find the utterance in $C$ that directly follows $\vec{a}$            ▷ We ensure $\vec{b}$ is not an end of conversation token.
3: $\vec{d} \leftarrow min(dist(\vec{b}, \vec{r}))$
4: return $1.0 - \frac{\vec{g}}{\|\vec{g}\|}$
```

Scattershot The key insight in the scattershot approach is the idea that there isn't really a 'single right answer' when it comes to utterance/response pairs. Instead, there exists a manifold (or perhaps multiple manifolds) of correct responses, any of which is equally valid. For example, the question "Have you seen Black Panther?" Can be answered with any of ['Yes', 'No', 'I'm going to watch it tonight', 'Are you kidding? Of course I've seen Black Panther']. From a strictly text-based perspective, all responses are equally valid. (You'd need a knowledge graph/self-model to determine which of the valid responses are also truthful.) Accordingly, the scattershot algorithm assumes that as long as you're close to *any one* of the valid responses, you've probably got a pretty good candidate.

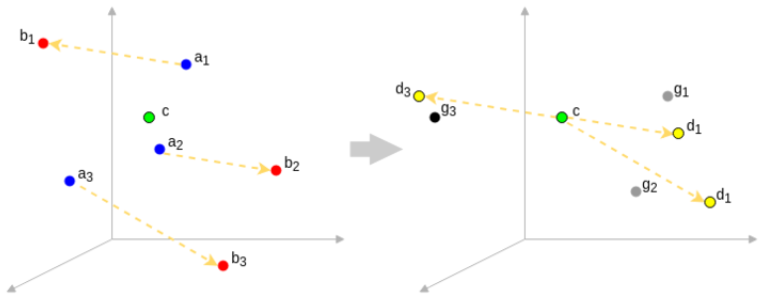

Figure 4: Scattershot Method, where: c (green) represents the embedded utterance, a_i (blue) represent the nearest embedded utterances from the Chit-Chat Dataset, b_i (red) represent the associated embedded response to a_i in the Chit-Chat Dataset, d_i (yellow) represent the 'ideal' responses, and g_i (grey and black) represent embedded responses generated by our system with g_3 (black) representing the response selected by the Scattershot ranking strategy.

Algorithm 5 Scattershot

```
Inputs:
$\vec{h}$ = Embedded conversation history.
$\vec{r}$ = Embedded candidate responses produced by the generators.
$\underset{n\times\|h\|-1}{C}$ = Embedded Chit-Chat dataset, where the columns are pairwise differences between subsequent utterances
Output:
$S = \{s_1 \dots s_i \mid s \in [0,1]\}$ where $s_i$ is the score for $r_i$

1: $\vec{c} \leftarrow [h_1 - h_0, \dots, h_n - h_{n-1}]$
2: for $i$ in 1..5 do                                    ▷ Find the $n$ closest points in $C$ to $c$.
3:     $\vec{a_i} \leftarrow min_i(dist(\vec{u}, C))$    ▷ Where $dist$ is any valid distance metric.
4:     $\vec{b_i} \leftarrow$ Find the utterance in $C$ that directly follows $\vec{a_i}$
5:     $\vec{d_i} \leftarrow \vec{b_i} - \vec{a_i} + \vec{c}$    ▷ Where $d_i$ is the "ideal" response vector to $\vec{b_i}$.
6: end for
7: for $r_i$ in $\vec{r}$ do
8:     $\vec{g_i} \leftarrow min(dist(\vec{r_i}, \vec{d}))$
9: end for
10: return $1.0 - \frac{\vec{g}}{\|\vec{g}\|}$
```

Flow Vectors The flow vectors approach is based on the idea that conversations tend to "flow" from certain regions of embedding space into others, and that all matching utterance pairs will reflect the same general flow direction. Rather than using a single conversation vector, this method averages multiple vectors and looks for a candidate utterance that matches the resulting flow direction.

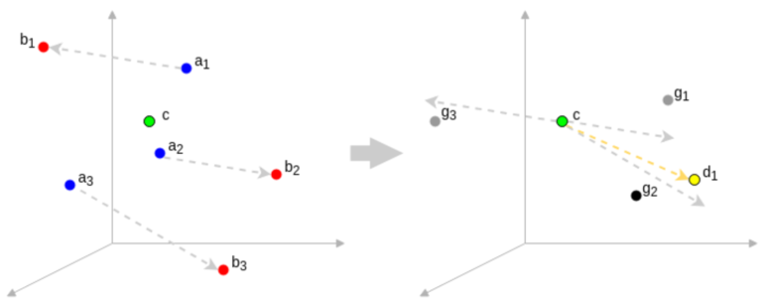

Figure 5: Flow Vectors Method, where: c (green) represents the embedded utterance, a_i (blue) represent the nearest embedded utterances from the Chit-Chat Dataset, b_i (red) represent the associated embedded response to a_i in the Chit-Chat Dataset, d_1 (yellow) represents the 'ideal' response, and g_i (grey and black) represent embedded responses generated by our system with g_2 (black) representing the response selected by the Flow Vectors ranking strategy.

Algorithm 6 Flow vectors

```
Inputs:
$\vec{h}$ = Embedded conversation history.
$\vec{r}$ = Embedded candidate responses produced by the generators.
$\underset{n\times\|h\|-1}{C}$ = Embedded Chit-Chat dataset, where the columns are pairwise differences between subsequent utterances
Output:
$S = \{s_1 \dots s_i \mid s \in [0,1]\}$ where $s_i$ is the score for $r_i$

1: $\vec{c} \leftarrow [h_1 - h_0, \dots, h_n - h_{n-1}]$
2: for $i$ in 1..5 do                                    ▷ Find the $n$ closest points in $C$ to $c$.
3:     $\vec{a_i} \leftarrow min_i(dist(\vec{u}, C))$    ▷ Where $dist$ is any valid distance metric. We used Euclidean distance.
4:     $\vec{b_i} \leftarrow$ Find the utterance in $C$ that directly follows $\vec{a_i}$.
5: end for
6: $\vec{\mathcal{F}} = 1/n \sum_{i=1}^{n} b_i - a_i$
7: $\vec{d} = \vec{a} + \vec{\mathcal{F}}$              ▷ Where $\vec{d}$ is the "ideal" response vector to $\vec{c}$.
8: $\vec{g} \leftarrow min(dist(\vec{r}, \vec{d}))$    ▷ Where $dist$ is any valid distance metric. We used Euclidean distance.
9: return $1.0 - \frac{\vec{g}}{\|\vec{g}\|}$
```

Table 1 shows examples of two closely related queries and a set of human-generated candidate responses. Notice that the two examples accurately distinguish the idea of being "from" or growing up somewhere from the idea of simply being born somewhere. In table 1a, candidates that capture the

Candidate responses	Distance
I am from California.	0.789
California.	0.843
Cali.	0.845
I grew up in Santa Fe but my family just moved to Salt Lake City.	0.855
I grew up in California.	0.864
NY.	0.899
New York.	0.925
Manhattan.	0.998
Yes.	1.058
Have you ever been to California?	1.082
I love Star Wars.	1.112
Babies are usually born in a hospital.	1.203

(a) Query: "Where are you from?"

Candidate responses	Distance
California.	1.065
Cali.	1.106
NY.	1.131
New York.	1.132
Manhattan.	1.158
I am from California.	1.18
I grew up in California.	1.21
I grew up in Santa Fe but my family just moved to Salt Lake City.	1.231
Have you ever been to California?	1.277
Babies are usually born in a hospital.	1.375
Yes.	1.394
I love Star Wars.	1.487

(b) Query: "Where were you born?"

Table 1: Flow vectors examples.

idea of growing up somewhere, like "I am from California" or "I grew up in California", are ranked higher (lower distance), while table 1b ranks simple locations higher. Both examples rank irrelevant responses like "I love Star Wars" and "Yes." lower.

Neural Network As a baseline comparison, we implemented a multi-layer regression network using Tensorflow [1]. Its input is two utterances from the Chit-Chat dataset (see Section 5.2), each embedded as a 512 dimensional vector using the Universal Sentence Encoder [4]. These are concatenated to produce a 1024 dimensional input vector. The output is a single 512 dimensional vector prediction of the best response. We used two hidden layers with 2048 and 2014 units respectively, with exponential linear unit (ELU) activation functions and a Mean Squared Error (MSE) loss function. We found that training the network using a simple stochastic gradient descent optimizer with a learning rate of .001 worked best. The dropout value was 25%.

5.3.1 Comparison

offline testing In order to evaluate the performance of the scaffolding algorithms in offline tests, we created a hand-annotated dataset taken from the bot's dialog history. For each user utterance, an Alexa Prize team member selected the best candidate response by hand. Because this was a time-consuming process, we contented ourselves with a small initial evaluation set of 630 examples. We then measured the performance of each algorithm variant on our evaluation set. Results are shown in Table 2. In offline testing, we found that the neural network was the most robust algorithm.

online testing When deployed in production, use of the Flow Vectors algorithm created the highest level of customer satisfaction, with an average rating of 3.25 (see Table 2). The disparity between these results and the offline tests suggests two things: (1) The combination of the Flow Vectors algorithm with the Chit-Chat scaffolding corpus was able to generalize more effectively than the neural network to the scope and variability of real-world human conversation. (2) The Flow Vectors method, while less accurate in offline tests, has the advantage of failing with dignity in production settings. In other words, it does not reliably select the 'optimal' response, but the responses it chooses instead are satisfactory to customers. Further research is required to quantify this phenomenon.

data density The three scaffolding algorithms implemented a failsafe mechanism when $\vec{b}$ landed in a sparse region of the Chit-Chat corpus. If the algorithm could not find enough nearby points to determine a good $\vec{g}$, it returned a value of 0, indicating it could not determine a score. The raw distances were still returned as an indication of confidence. The neural network, which did not have this failsafe mechanism, was especially problematic because it simply guessed during inference when it found itself in a sparse region. During offline testing, data sparsity errors occurred approximately 25% of the time.

Offline	Number Correct	Accuracy
Naïve	333	52.8571%
Scattershot	337	53.4921%
Flow vectors	327	51.9048%
Neural network	402	63.8095%

Online	Avg rating	Conversations
Naïve	3.14	177
Scattershot	3.11	180
Flow vectors	3.25	275
Neural network	2.99	214
No algorithm	2.99	204

Table 2: Left: Accuracy of scaffolding algorithms in offline tests. To count as correct, the ideal answer must be in the top 40% of the algorithm's preferred responses. Right: Average rating received during A/B testing. Tests were performed between 8/18/2018 and 8/24/2018 inclusive, with each conversation randomly assigned to use one of the scaffolding variants, the neural network predictor, or no scaffolding algorithm at all. All other system parameters were held constant.

6 Supporting Infrastructure

Our response generators rely on preprocessing and keyword/entity extraction. These tasks are performed by the system infrastructure, which also handles scalability, parallel processing, response ranking, and so forth. An overview of this infrastructure is provided below.

6.1 Topic Detection and Entity/Keyword Extraction

Topic detection is accomplished using Amazon's topic classifier, which usually runs in 600 ms or less and provides reasonably good accuracy. Entity detection is accomplished via a combination of Spacey's entity detection capabilities and Google's natural language processing API. Keywords are extracted using hand-coded text analysis; for example if the word 'about' appears in the user utterance, the words immediately following it are assumed to be keywords. All entities and keywords are filtered to prevent the agent from processing inappropriate language. More information about the filtering process is provided in section 6.4.

6.2 Offensive Speech Detection

It is vital that we prevent the socialbot from using language that could be considered inappropriate, and it would be useful to know when the users are using inappropriate language so that we can approach the conversation carefully. Amazon has a built-in offensive speech classifier, but it isn't as fast as a homemade solution.

We started with the hand-annotated blacklist from Amazon's sample code, which consisted of a single long list of offensive words. Language is not black and white though, and we needed gray areas; there are some topics that some users might not want to hear about, while other users actively try to discuss them. We divided the blacklist into three different categories: 'severe', 'medium', and 'mild'. The 'severe' category is for words that we never want the bot to say. This list is comprised of words that are offensive alone, without context. The 'medium' category is for words that usually occur in the context of offensive language. The 'mild' category is reserved for words that indicate possibly offensive or inappropriate topics such as 'murder' and 'guns'. We do not want our socialbot engaging with these topics without user permission, and possibly not at all. Yet for some users, 'guns' are a perfectly acceptable conversation topic, and we wanted our code to reflect that.

6.3 Intent Recognition

We tested various methods for intent recognition, including neural networks and the use of distance metrics within sentence-level embedding spaces to classify utterances. In the end, we found that a hand-coded solution using a priori knowledge about conversational patterns was most effective. Our hand-coded intent recognizer is capable of triggering more than one intent at a time. For example, if the user said "oh yes that is wonderful", the intent recognizer would identify "oh" as "useless", "yes" as "yes response", and "that is wonderful" as "positive". This is accomplished using simple regular expressions, so there is nothing particular novel being done. That said, it works remarkably well, provided most cases of user speech can be anticipated by the system's designers.

Future work on intent recognition could include the use of embedding grammars to generalize from a small set of hand-crafted expressions to a wide array of synonyms and related terms [35]. Preliminary experiments suggest that this would allow the system to extract intents from utterances that would otherwise remain unlabeled.

6.4 Response Evaluation (Ranking & Filtering)

In a system with many generators, a "best response" must be chosen from among the generated candidates. Combined with the current state of the conversation, all generated response sentences must be evaluated for consistency and relevance. This evaluation should also take into account user personality and the system's synthetic personality elements.

During the design process, we decided that the function that selects the final response to be spoken by the system (hereafter known as the Arbiter) should accept as input a matrix containing ranked features about each possible output response. This function would evaluate the features of each sentence, rank the sentences, and return a concatenation of several highly-rated responses.

Response Ranker List

1. **Confidence:** Each response generator hands the ranker a confidence rating, which is a measure of the value the generator places on its generated response. This requires a division of labor, as well as some evaluative code in each generator.
2. **Context:** Some response generators are better equipped to handle certain topics than others. The context ranker examines the topic of the user utterance to determine how well-equipped each generator is to address the current conversation state. For example, when the current topic is "movies", the movie response generator will receive a higher score.
3. **Embedding Distance:** A conversational scaffolding method (see Section 5.1) is used to evaluate how well each candidate response matches the "flow" of the current conversation. Along with the Context and Preference rankers, this is one of the system's most heavily-weighted factors, as shown in Figure 8.
4. **Intent:** This is a measure of how well a response matches the intent triggered by the user's input sentence. The intents are shown in Figure 6.

Intent	Brief Description
yes response	affirmations
no response	negations
imperative	command words
opinion request	question specifically about opinions
general fact request	question regarding general knowledge
name info	giving us their name
user fact	offering information about themselves
song request	sing a song
rap request	rap
meta request	question about the conversation
hard stop	stop talking right now
soft stop	say goodbye and end the conversation
opening request	general small talk
apathetic	apathetic response
emotional cause (and effects)	user describes something
emotional effect (and causes)	user describes a state he's in

Figure 6: A list of the created intents and a brief description of each one.

Once the intents have been recognized in the user input, the intent recognizer is run on each of the generated responses, and we use the comparison between the intents found in those sentences to identify whether the responses have a compatible intent (see Figure 4). As an example, if the user input is found to contain the "apathetic" intent, the generated response is said to match if it contains the "opinion request", "general fact request", or "user fact" intent. If a user is not currently engaged, the system will try to engage them with facts or questions. Below in Figure 7 we show the entire mapping of matched intents.

5. **Keyword Model:** Measuring the state of a human's personality is difficult, but we can measure certain aspects of their personality and try to tailor our responses to their mood and

User Intent	Response Intent	Match
opinion request	user fact	yes
name info	opening request	yes
user fact	positive, opinion request, general fact request, meta request	yes
soft closing request	soft closing request	yes
opening request	opening request	yes
apathetic	opinion request, general fact request, user fact	yes
emotional cause	emotional effect	yes
emotional effect	emotional cause	yes
imperative	yes response, no response, opinion request, general fact request	no

Figure 7: A list of user intents and response intents paired to indicate either a positive or negative match.

interests. This would cause EVE to have more personalized conversations with individuals, making for a more interesting experience. The keyword model tries to detect the current mood/emotion of the user, and use that to upvote certain response generators. If it detects that the user 'wants to learn', fact-based response generators will score more highly.

6. **Length:** The length of the response is critical in influencing user engagement. In early user trials, we observed that if the system talks for too long at a time, the user becomes less engaged with the socialbot. But this also works in reverse; if the socialbot says too little, the user also disengages. After watching several users interact with our system in real-time, we calibrated the length ranker to rank a given sentence as ideal when it's between seven and 15 words. This is under the assumption that we will be concatenating multiple responses to create a final output with a target length between 15 and 25 words. Our socialbot will never return a response comprised of more than 30 words.
7. **Preference:** As the system's developers, we trust certain response generators to generally answer better than others. We added the preference ranker to give individual preference to the most sophisticated and well-coded generators. Each generator is assigned a preference rank between zero and one, and that is handed to the Arbiter as a feature and is factored into the final decision.
8. **Repeat:** No one wants to hear the same sentences over and over, and no human communicates that way. The repetition ranker flags sentences that have been offered to the user in the last 15 conversation steps. This feature is essential to ensure a good user experience with each new response from the socialbot.
9. **Sentiment:** The sentiment ranker seeks to emphasize positive responses while still being respectful of the user's mood. We use Google's sentiment classification API to evaluate the sentiment of each sentence.

The Arbiter's task is to accept a feature matrix comprised of scores from the various rankers. Each feature in the matrix has an assigned scalar which is used to temper or enhance its influence in deciding the final ranking of the responses. The final ranking is the sum of the Hadamard product of the features and their respective scalars, multiplied by a throttle factor composed of the score from the throttling ranker (repetition) and its respective scalar:

$$(repetition\_ranking * 2.0) * \sum_{i=1}^{9} feature\_vector_i * scalars_i$$

It's worth noting that this equation shows a feature vector (as opposed to a matrix), because here we're showing the Arbiter running on only a single response for clarity. It's also important to note that the 'feature_vector' referenced in the equation doesn't contain the throttling feature.

6.5 Response Combination

Once the Arbiter has produced a ranking for each candidate response, the responses are combined into an emote/answer/offer framework as described in Section 1.4. This is accomplished by using a sequence of hand-coded rules including the following:

Feature	Scalar
Confidence	0.5
Context	3.0
Embedding Distance	2.0
Intent	1.0
Keyword Model	1.0
Length	0.5
Preference	4.0
Repetition	2.0
Sentiment	0.5

Figure 8: A list of the features and their respective scalars.

1. The highest-ranked answer response is always used.
2. If an emotive response with a sufficiently high ranking exists, prepend it to the answer.
3. If the answer response does not end in a question mark or other conversational prompt, and if an offer response with a sufficiently high ranking exists, append the highest-ranked offer to the answer.

Additional rules ensure that the final response is not too long and that system does not combine two responses which are substantially similar in content. Certain response generators, such as MovieBot, generate responses that always include both an answer and an offer; no offers are ever appended to responses from these generators. Additionally, conversational 'stitching' is sometimes performed by removing the prefixes ['Okay', 'Sure!', 'So'] from answer responses when an emote is prepended.

7 Future Work

7.1 Improved Knowledge Graph Structure and Traversal

Moving forward, we wish to expand the number and types of edges detected by DataBot and utilized by ModelqueryBot. In addition to 'called', 'likes', and 'dislikes' edges, we aim to explore methods that might automatically identify and curate a list of important edges to include in the entity models.

Our team's near-field work is also focused on expanding the system functionality to model other entities besides the user and the system itself. As an example, if the user says "I really liked (movie title) because (name of actor) was hilarious.", it would be fairly straightforward to generate a user-specific node for '(name of actor)' and record that the actor is funny. The system would only remember this 'fact' when interacting with the specific user who provided the information, and if the system interacted at a later point in time with a user who disliked the same actor the system would adapt its knowledge graph model of that actor to reflect the negative sentiment of the new user. In this way, the system could remember to avoid topics that the current user doesn't enjoy and direct the conversation towards topics the user finds more appealing.

By discussing particular topics with the system, the user will be able to mold the personality of the system to themselves; future work aims to enable the system itself to alter its own personality dynamically by incorporating probabilistic modeling into the user and self models. Using statistics on the models gleaned across thousands of interactions with users, probabilistic modeling could enable the system to infer unspoken opinions users might hold and the types of utterances the system should produce given the current state of its models.

7.2 Optimized Embeddings Module

While our conversational scaffolding techniques show valuable potential, much work remains in order to determine the optimal algorithm for prioritizing candidate responses. We are in the process of evaluating the effect of scaffolding algorithm, distance threshholds, and conversation history on classification accuracy in offline tests.

7.3 Similarity Metrics for Data Retrieval

Scaffolding is only one of many possible uses for universal sentence embeddings. In the future, we plan to deploy a data retrieval mechanism that will allow response generators to customize their information to specific user utterances. This can be done by measuring the distance (in embedding space) between the user utterance and each candidate response. For example, if the user says "Tell me news about 12 people trapped in a mine", a nearest-neightbor search in embedding space would likely return headlines such as "Twelve boys trapped in a Thai Cave, Crews working to rescue them", a trending news topic from July 2018. Preliminary experiments suggest that this method is more effective than a keyword search (which would have matched only the words 'trapped in a').

A similar approach can be used to improve the ability of response generators to correctly identify which knowledge graph node was invoked by the user's utterance. As described in Section 2.2, each node in the knowledge graph is accompanied by a universal sentence embedding corresponding to the node's description. This embedding can be used to disambiguate between identically-labeled nodes.

For example, if the user says "I watched Harry Potter this weekend", it can be difficult for our knowledge graph algorithms to determine whether the user is referring to the "Harry Potter" node that describes a series of books, the "Harry Potter" node that describes a movie series, or the "Harry Potter" node that describes the main character in both. Selecting the wrong node can result in jarring conversational disconnects. However, since previous utterances in the conversation were likely focused on movies (and because the current utterance utilizes movie-related phrases like 'watched' and 'weekend') the average distance of the preceding sentences to the embedded description of each knowledge graph node will likely reveal the correct interpretation of the user's meaning.

7.4 ScriptDog: A Language for Managing Conversational State

One challenge inherent in dialog management is tracking conversational state. As part of the competition, we developed a language specifically built to describe complex, factored dialog scripts with a natural, python-like syntax. The language is called "ScriptDog", for "scripted dialog" [33].

The primary feature of the language is the ability to store the program state in a JSON-serializable object. This makes it easy to, for example, stash the program state in persistent storage and reload it at a later time. (This is particularly useful when program execution is broken up across multiple invocations that are distributed across multiple machines). The model makes it easy to integrate with AWS Lambda and other serverless computing frameworks. Tight integration with python makes it possible to cleanly separate conversational state from the backend logic that drives transitions, such as integration with databases, NLP processors, knowledge graphs, etc.

Other key language features include (1) factored, reusable state sequences; (2) factored, reusable transition definitions; (3) direct language support for random choices; (4) tight integration with regular expressions; (5) global transitions that are defined once, but implicitly accessible at any point. Future response generators for EVE will use the ScriptDog language and its associated features.

8 Conclusion

BYU-EVE is an open domain dialogue architecture founded on the principle that centralized knowledge representation coupled with self- and user-modeling can produce dynamic, human-like conversations. The key innovations in this research are twofold: (a) we treat conversation as a multi-channel information stream along which not only information, but also emotive and psychological data is being passed, and (b) we combine structured knowledge graph traversal with deep learning in order to produce fluid, personable responses that are grounded in known facts.

As part of Amazon's Alexa Prize Challenge, EVE remains a work in progress. Our next research steps include improved knowledge graph traversal, optimized methods for conversational scaffolding, and dynamic text generation via neural networks conditioned on knowledge graph nodes. Our goal of achieving truly open domain, free-form conversation is ambitious, but it is also presents intriguing possibilities. If it can be achieved, it will open doors for general purpose, responsive conversational systems that require far less retrieval and far less hand-coding than the present state-of-the-art. The resulting systems would positively impact fields including education, personal assistants, elder care, and many others.

References

[1] 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. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Davide Alocci, Julien Mariethoz, Oliver Horlacher, Jerven T. Bolleman, Matthew P. Campbell, and Frederique Lisacek. Property graph vs rdf triple store: A comparison on glycan substructure search, December 2015.

[3] Kevin K Bowden, Shereen Oraby, Amita Misra, Jiaqi Wu, Stephanie Lukin, and Marilyn Walker. Data-driven dialogue systems for social agents. In *Advanced Social Interaction with Agents*, pages 53–56. Springer, 2019.

[4] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder. *CoRR*, abs/1803.11175, 2018.

[5] Cristian Danescu-Niculescu-Mizil and Lillian Lee. Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs. In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics, ACL 2011*, 2011.

[6] Souripriya Das, Jagannathan Srinivasan, Eugene Inseok, and Jayanta Benerjee. A tale of two graphs: Property graphs as rdf in oracle, 2014.

[7] The Internet Movie Script Database. The internet movie script database.

[8] Ahmed Fadhil. Beyond patient monitoring: Conversational agents role in telemedicine & healthcare support for home-living elderly individuals. *arXiv preprint arXiv:1803.06000*, 2018.

[9] Python Software Foundation. gremlinpython 3.3.3, May 2018.

[10] The Apache Software Foundation. Apache tinkerpop, April 2015.

[11] Nancy Fulda, Danieal Ricks, Ben Murdoch, and David Wingate. What can you do with a rock? affordance extraction via word embeddings. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 1039–1045, 2017.

[12] Nancy Fulda, Nathan Tibbetts, Zachary Brown, and David Wingate. Harvesting common-sense navigational knowledge for robotics from uncurated text corpora. In *Proceedings of the First Conference on Robot Learning (CoRL) - forthcoming*, 2017.

[13] Christian Grimme, Mike Preuss, Lena Adam, and Heike Trautmann. Social bots: Human-like by means of human control? *Big data*, 5(4):279–293, 2017.

[14] Ryuichiro Higashinaka, Kenji Imamura, Toyomi Meguro, Chiaki Miyazaki, Nozomi Kobayashi, Hiroaki Sugiyama, Toru Hirano, Toshiro Makino, and Yoshihiro Matsuo. Towards an open-domain conversational system fully based on natural language processing. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 928–939, 2014.

[15] Ben Krause, Marco Damonte, Mihai Dobre, Daniel Duma, Joachim Fainberg, Federico Fancellu, Emmanuel Kahembwe, Jianpeng Cheng, and Bonnie L. Webber. Edina: Building an open domain socialbot with self-dialogues. *CoRR*, abs/1709.09816, 2017.

[16] Pavel Kucherbaev, Achilleas Psyllidis, and Alessandro Bozzon. Chatbots as conversational recommender systems in urban contexts. In *Proceedings of the International Workshop on Recommender Systems for Citizens*, page 6. ACM, 2017.

[17] Ser Ling Lim and Ong Sing Goh. Intelligent conversational bot for massive online open courses (moocs). *CoRR*, abs/1601.07065, 2016.

[18] H. Liu and P. Singh. Conceptnet — a practical commonsense reasoning tool-kit. *BT Technology Journal*, 22(4):211–226, October 2004.

[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[20] Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. Association for Computational Linguistics, May 2013.

[21] Film Scripts Online. Film scripts online.

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

[23] Bastian Quilitz and Ulf Leser. Querying distributed rdf data sources with sparql. In Sean Bechhofer, Manfred Hauswirth, Jorg Hoffmann, and Manolis Koubarakis, editors, *The Semantic Web: Research and Applications*, pages 524–538, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[24] Geetanjali Rakshit, Kevin K Bowden, Lena Reed, Amita Misra, and Marilyn Walker. Debbie, the debate bot of the future. In *Advanced Social Interaction with Agents*, pages 45–52. Springer, 2017.

[25] Ashwin Ram, Rohit Prasad, Chandra Khatri, Anu Venkatesh, Raefer Gabriel, Qing Liu, Jeff Nunn, Behnam Hedayatnia, Ming Cheng, Ashish Nagar, Eric King, Kate Bland, Amanda Wartick, Yi Pan, Han Song, Sk Jayadevan, Gene Hwang, and Art Pettigrue. Conversational AI: the science behind the alexa prize. *Alexa Prize Proceedings*, abs/1801.03604, 2018.

[26] Reddit. Reddit datasets.

[27] Anna Rogers, Aleksandr Drozd, and Satoshi Matsuoka. Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn't. *Proceedings of the NAACL Student Research Workshop*, pages 8–15, 01 2016.

[28] Hassan Saif, Miriam Fernandez, Yulan He, and Harith Alani. Evaluation datasets for twitter sentiment analysis. a survey and a new dataset, the sts-gold, 12 2013.

[29] Heung-yeung Shum, Xiao-dong He, and Di Li. From eliza to xiaoice: challenges and opportunities with social chatbots. *Frontiers of Information Technology & Electronic Engineering*, 19(1):10–26, 2018.

[30] Denny Vrandecic and Markus Krotzsch. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, September 2014.

[31] Zihao Wang, Ali Ahmadvand, Jason Ingyu Choi, Payam Karisani, and Eugene Agichtein. Emersonbot: Information-focused conversational ai emory university at the alexa prize 2017 challenge. *AWS*, 2017.

[32] Tsung-Hsien Wen, David Vandyke, Nikola Mrksic, Milica Gasic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*, 2016.

[33] David Wingate and Tyler Etchart. Scriptdog. `https://github.com/BYU-PCCL/scriptdog/`, 2018.

[34] David Wingate, William Myers, Tyler Etchart, and Nancy Fulda. BYU 2018 chit chat challenge. `https://chitchatchallenge.com/`, April 2018.

[35] David Wingate, William Myers, Nancy Fulda, and Tyler Etchart. Embedding Grammars. *ArXiv e-prints*, August 2018.