The field of conversational AI is rapidly evolving; we alone reference eight papers written within the last year. However, today's proposed designs often limit their scope to a single neural network architecture. Each proposal attempts to utilize one solution for the given problem. Few proposals have brought multiple architectures together in a way that can provide new, robust response and behavior. Our goal is to bring together state-of-the-art Neural Network (NN) designs to achieve more coherent and deeper conversations.

Since we do not know a priori how these architectures interact with each other, we must develop a methodology to test various permutations of them. Below, we will discuss possible methodologies for determining designs that interface well with each other. Additionally, we will discuss different implementations and examine how they fit into the architecture proposed in our design document.

For each module in the design document we consider various alternative implementations. For example, DMNs could be replaced with database queries, and LSTMs could be substituted with TACNTNs (Yu Wu et al. https://arxiv.org/abs/1605.00090). Each of these modules would be trained on equivalent data and incorporated into the overall design to test their efficacy. Such training can be accomplished using reinforcement learning as proposed in Jiwei Li's paper Deep Reinforcement Learning for Dialogue Generation (Li et al. https://arxiv.org/abs/1606.01541). We plan to experiment with new rewards and different weights than Li et al.'s original heuristic.

Similar module implementations can also be interchanged intelligently if their response accuracy is similar. As their accuracy may be dependent on input sentence structures, keeping both architectures would be worthwhile with a specially trained classifier that sends inputs into either option. We would train the classifier to maximize overall accuracy of inputs by selecting one of these options. This input dispatch design could be kept in the final architecture or used solely to analyze weaknesses of each module to further refine training. Furthermore, this methodology can be applied to every component of our design when experimenting with the various low-level implementations outlined in the next section of this document.

Current examples of these architectural decisions in the DD section include topic classifications. There is a rich history of research in text topic classification using SVMs, LDA, and CNNs. Each of these models achieves unique results when tasked with analyzing different sentence styles as well as inferring and working around rare word usage. Choosing between classifiers would require experimenting with the specific phrases used in conversation rather than in written text. Should ambiguities arise we can rely on the topic database's reported confidence score to determine which of the topics most accurately represents and responds to the query.

After topic classification, the data flows to our memory bank. Within this memory bank, the classified query is dispatched to a relevant topic database, which could be a DMN+ bank as described in our design document or a custom database of relevant tuples based on Freebase. Furthermore, we will explore the possibility of storing local user topics as users communicate their topics of interest and preferences to our system. Assuming underlying DMN+'s, we can train these new topic-based memory units on relevant Wikipedia pages. Results generated by the Socialbot can be compared to sentences from the Wikipedia pages to gauge accuracy and facilitate training.

We extend the DCGM architecture by incorporating additional persona and user sentiment embeddings into our model. We propose representing each individual speaker as a vector that encodes background information and speaking style. These vectors can be learned jointly with word embeddings by back propagating word prediction errors during training. It is also possible to capture interaction patterns within conversation by linearly combining two speakers' vectors. A persona vector space would serve to cluster users by their traits without the need for explicit annotations. Similarly, we can also inject a sentiment vector into our hidden layer, which we can generate by learning sentiment embeddings within a vector space (http://www.aclweb.org/anthology/P08-2034) or by training a recursive auto-encoder on Stanford's Experience Project Dataset. By enriching our DCGM model accordingly, we will better predict personalized responses.

We also plan to experiment with various word embeddings in each module. We are currently considering Fasttext plus multiple layers of LSTMs to map synonyms together in our embedding (https://arxiv.org/abs/1607.01759v2 and https://arxiv.org/abs/1607.04606). The aim of this decision is to understand synonyms in the user's dialogue for referencing stored knowledge: terms such as "old" and "age" will be mapped close together, which boosts the performance of our topic modules and facilitates information retrieval. This method also implements a softmax activation function, which is compatible with the Pointer Sentinel Mixture Model (https://arxiv.org/abs/1609.07843). This model has demonstrated promising results in resolving the OOV problem (handling rare and unseen words). We also mention an alternate approach that uses TACNTNs (https://arxiv.org/abs/1605.00090) to take advantage of the inherent symmetries and structures of language since word order can be as important as word choice. However, without the PSMM, rare words may be more difficult to handle.

Our model assumes access to the most likely phrase from the Alexa Skill Kit. We intend to host our Socialbot on AWS Lambda in a general callback function that handles all responses given a "heard" text query. In the callback, our model will take in this user query and generate a text response for Alexa.