# Talking to Your TV: Context-Aware Voice Search with Hierarchical Recurrent Neural Networks

Jinfeng Rao,[1,2] Ferhan Ture,[1] Hua He,[2] Oliver Jojic,[1] and Jimmy Lin[3]
[1] Comcast Applied AI Research Lab
[2] Department of Computer Science, University of Maryland
[3] David R. Cheriton School of Computer Science, University of Waterloo
jinfeng@cs.umd.edu,ferhan_ture@comcast.com

## ABSTRACT

We tackle the novel problem of navigational voice queries posed against an entertainment system, where viewers interact with a voice-enabled remote controller to specify the TV program to watch. This is a difficult problem for several reasons: such queries are short, even shorter than comparable voice queries in other domains, which offers fewer opportunities for deciphering user intent. Furthermore, ambiguity is exacerbated by underlying speech recognition errors. We address these challenges by integrating word- and character-level query representations and by modeling voice search sessions to capture the contextual dependencies in query sequences. Both are accomplished with a probabilistic framework in which recurrent and feedforward neural network modules are organized in a hierarchical manner. From a raw dataset of 32M voice queries from 2.5M viewers on the Comcast Xfinity X1 entertainment system, we extracted data to train and test our models. We demonstrate the benefits of our hybrid representation and context-aware model, which significantly outperforms competitive baselines that use learning to rank as well as neural networks.

## 1 INTRODUCTION

Voice-based interactions with computing devices are becoming increasingly prevalent, driven by several convergent trends. The ubiquity of smartphones and other mobile devices with restrictive input methods makes voice an attractive modality for interaction: Apple's Siri, Microsoft's Cortana, and the Google Assistant are prominent examples. Google observed that there are more searches taking place from mobile devices than from traditional desktops [31], and that 20% of mobile searches are voice queries [23]. The success of these products has been enabled by advances in automatic speech recognition (ASR), thanks mostly to deep learning.

Increasing comfort with voice-based interactions, especially with AI agents, feeds into the emerging market on "smart homes". Products such as Amazon Echo and Google Home allow users to control a variety of devices via voice (e.g., "turn on the TV", "play music by Adele"), and to issue voice queries (e.g., "what's the weather tomorrow?"). The market success of these products demonstrates that people do indeed want to control smart devices via voice.

In this paper, we tackle the problem of navigational voice queries posed against an entertainment system, where viewers interact with a voice-enabled remote controller to specify the program (TV shows, movies, sports games) they wish to watch. If a viewer wishes to watch the popular series "Game of Thrones", saying the name of the program should switch the television to the proper channel. This is simpler and more intuitive than scrolling through channel guides or awkwardly trying to type in the name of the show on the remote controller. Even if the viewer knows that Game of Thrones is on HBO, finding the right channel may still be challenging, since entertainment packages may have hundreds of channels.

Our problem is challenging for a few reasons. Viewers have access to potentially tens of thousands of programs (including on-demand titles), whose names can be ambiguous. For instance, "Chicago Fire" could refer to either the television series or a soccer team. Even with recent advances, ASR errors can exacerbate ambiguity by transcribing queries like "Caillou" (a Canadian children's education television series) as "you". In addition, we observe that voice queries are very short, which makes the prediction problem more difficult because there is less signal to extract.

**Contributions.** We tackle the above challenges using two key ideas to infer user intent: hybrid query representations and modeling search sessions. Specifically, our contributions are as follows:

- To our knowledge, we are the first to systematically study voice queries in the entertainment context. We propose a technique to automatically collect ground truth labels for voice query sessions from real-world usage data.
- Our model has two key features: First, we integrate word- and character-level query representations. Second, we model voice search sessions to understand the contextual dependencies in query sequences. Both are accomplished with a probabilistic framework in which recurrent and feedforward neural network modules are organized in a hierarchical manner.
- Evaluations demonstrate the effectiveness of our hybrid query representation and context-aware models, significantly outperforming competitive baselines. Detailed analyses clarify *how* our models are better able to understand user intent.

## 2 BACKGROUND AND RELATED WORK

The context of our work is voice search on the Xfinity X1 entertainment platform by Comcast, one of the largest cable companies in the United States with approximately 22 million subscribers in 40 states. X1 refers to a software package distributed on top of Xfinity's most recent cable box, which has been deployed to 17 million customers


*CIKM'17, November 6–10, 2017, Singapore.*

ISBN 978-1-4503-4918-5/17/11…$15.00
DOI: http://dx.doi.org/10.1145/3132847.3132893

since around 2015. X1 can be controlled via the "voice remote", which is a remote controller that has an integrated microphone to receive voice queries from viewers. The current deployed system is based on a combination of hand-crafted rules and machine-learned models to arrive at a final response. The system has a diverse set of capabilities, which increases query ambiguity and magnifies the overall challenge of understanding user intent. These capabilities range from channel change to entity search (e.g., sports team, person, movie, etc.). In addition, voice queries may involve general questions, from home security control to troubleshooting the wifi network, or may be ultimately directed to external apps such as Pandora. In this paper, we focus on navigational voice queries where viewers specify the TV program they wish to watch.

In our application, we receive as input the one-best transcription from the ASR system. We do not have access to the acoustic signal, as the ASR system is a black box. While it would be ideal if we could build joint models over both the acoustic signal, transcription lattice, and user intent, in many operational settings this is not practical or even possible. In the case of X1, the ASR is outsourced to a third party—a scenario not uncommon in many organizations. Thus, transcription errors compound ambiguity in the queries and make our problem harder.

We are, of course, not the first to tackle voice search [1, 4, 6, 26, 34], although to our knowledge we are the first to focus on voice queries directed at an entertainment system. How is this particular domain different? The setting is obviously different—in our case, viewers are clearly sitting in front of a television with an entertainment intent. To compare and contrast viewers' actual utterances, we can turn to previously-published work that studied the characteristics of voice search logs, especially in comparison to text search data [5, 10, 25, 35]. Schalkwyk et al. [25] reported statistics of queries collected from Google Voice search logs, which found that short queries, in particular 1-word and 2-word queries, were more common in the voice search setting, while long queries were much rarer. In contrast, a more recent study by Guy [10] reported that voice queries tend to be longer than text queries, based on a half-million query dataset from the Yahoo mobile search application. The average length across 32M voice queries is 2.04 in our dataset, much shorter than the reported average of 4.2 for Yahoo voice search [10].

We note another important difference between our entertainment context and voice search applications on smartphones: on a mobile device, it is common to back off to a web search if the query intent is not identified with high confidence. Less than half of Yahoo voice queries (43.3%) are handled by a pre-defined card [10]. While we are not aware of a scientific study about such behavior on a TV, our intuition is that a list of search results is less useful to TV viewers than it might be for smartphone users, since subsequent interactions are more awkward: it is difficult for users to scroll and they have limited input methods for follow-up interactions. Of course, this does not mean that ranking is unimportant, and we evaluate our methods using several rank-based metrics.

Research on voice query reformulations is also relevant to our work on session modeling. Jiang et al. [14] analyzed different types of ASR errors and users' corresponding reformulation strategies. Hassan et al. [11] built classifiers to detect query reformulation. Shokouhi et al. [27] discovered that users do not tend to switch between voice and text when reformulating queries. A more recent paper [28] proposed an automatic way to label voice queries by examining post-click and reformulation behaviors, which produced a large amount of "free" training data to reduce ASR errors. These papers provide a source of inspiration for our models.

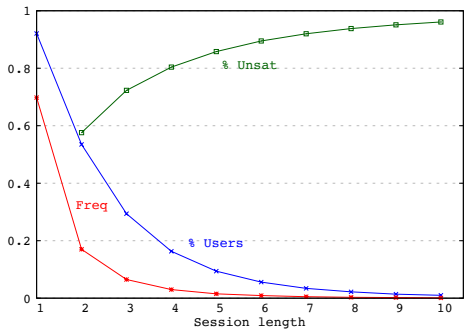



**Figure 1: For each session length, we plot three values: frequency of sessions (red), percentage of users that issued at least one session of that length (blue), and percentage of users with at least one unsatisfactory experience (green).**

Our approach to tackling ambiguous voice queries is to take advantage of context in voice search sessions. Our assumption is that when a viewer is not satisfied with the results of a query, she will issue more queries in rapid succession and continue until the desired program is found or until she gives up. In fact, we've found that across 20M sessions in a week (details in Section 4.1), more than 30% of sessions have multiple queries, accounting for over 57% of all queries. Figure 1 shows that more than half of the users issued at least one multi-query session in that week.

Preliminary explorations suggest that multi-query sessions are at least in part due to an unsatisfied user reformulating the original query. That is, sessions are long because users couldn't find what they were looking for on the first try. Since our deployed system does not have interaction or dialogue capabilities, we need a proxy for measuring user satisfaction. For each session in our voice query log (details in Section 4.1), we computed the character edit distance between every pair of queries (normalized by the sum of query lengths) and labeled the session "unsatisfactory" if the minimum was less than 25% (two identical queries will have a zero edit distance), the intuition being that reformulated queries are likely to be similar. To verify, we sampled some of these sessions and confirmed that our heuristic does indeed capture reformulation-heavy sessions. In our dataset, over half of the users had at least one voice session with multiple queries (blue line in Figure 1), and among these users, close to 60% of them had at least one unsatisfactory experience (green line). This rate keeps going up as the session length increases.

Initial analyses suggest that better modeling of context in multi-query sessions may lead to an improved user experience. For example, compare two sessions: ["ncis", "cargo fire", "chicago fire"] and ["espn", "chicago sports", "chicago fire"]. Although both end in the same query, it is fairly clear that in the first case, the viewer is interested in the TV drama series "Chicago Fire" (previous queries all mention other drama series), whereas in the latter, the viewer is interested in the sports team with the same name. Modeling

context in search sessions is of course not a new idea, and our approach is inspired by previous work in web search [2, 3, 8, 16–18, 30]. However, we are working in a completely different domain. Building on recent advances in deep learning applied to information retrieval [20, 21, 24, 33], we decided to implement our ideas using neural networks.

# 3 MODEL ARCHITECTURE

## 3.1 Problem Formulation

Given a voice query session $[q_1, \ldots, q_n]$, our task is to predict the program $p$ that the user intends to watch. We perform this prediction cumulatively at each time step $t \in [1, n]$ on each successive new voice query $q_t$, exploiting all previous queries in the session, $[q_1, \ldots, q_{t-1}]$. For example, in a three-query session $s_i = [q_{i_1}, q_{i_2}, q_{i_3}]$, there will be three separate predictions: first with $[q_{i_1}]$, second with $[q_{i_1}, q_{i_2}]$, and third with $[q_{i_1}, q_{i_2}, q_{i_3}]$. We sessionize the voice query logs heuristically based on a time gap (in this case, 45 seconds—more details later), similar to how web query logs are sessionized based on inactivity. As described above, each query is a text string, the output of a third-party "black box" ASR system that we do not have internal access to.

We aim to learn a mapping function $\Theta$ from a query sequence to a program prediction, modeled using a probabilistic framework:

$$\begin{aligned} &\text{Data: } D = \{(s_i, p_i) \mid s_i = [q_{i_1}, ..., q_{i_{|s_i|}}],\ p_i \in \Phi\}_1^{|D|} \\ &\text{Model: } \hat{\theta} = \arg\max_{\theta} \prod_{i=1}^{|D|} \prod_{t=1}^{|s_i|} P(p_i | q_{i_1}, ..., q_{i_t}; \theta) \end{aligned} \tag{1}$$

where $D$ denotes a set of labeled sessions ($s_i$ denotes the $i$-th session with $|s_i|$ queries), $p_i$ is the intended program for session $i$, $\Phi$ is the global set of programs, and $\theta$ is the set of parameters in the mapping function $\Theta$. Our goal is to maximize the product of prediction probabilities.

We decompose the program prediction task into learning three mapping functions: a query embedding function $\mathbb{F}(x; \theta_{\mathbb{F}})$, a contextual function $\mathbb{G}(x; \theta_{\mathbb{G}})$, and a classification function $\mathbb{H}(x; \theta_{\mathbb{H}})$. The query embedding function $\mathbb{F}(\cdot)$ takes the text of the query as input and produces a semantic representation of the query. The contextual function $\mathbb{G}(\cdot)$ considers representations of all the preceding queries as context and maps them to a high-dimensional embedding vector to capture both semantic and contextual features. Finally, the classification function $\mathbb{H}(\cdot)$ predicts possible programs from the learned contextual vector. We adopt the following decomposition:

$$\begin{aligned} P(p_i | q_{i_1}, ..., q_{i_t}) \sim P(p_i | c_{i_t}) \cdot P(c_{i_t} | v_{i_1}, ..., v_{i_t}) \\ \cdot P(v_{i_1}, ..., v_{i_t} | q_{i_1}, ..., q_{i_t}) \end{aligned} \tag{2}$$

where $c_{i_t}$ denotes the *contextual* embedding of the first $t$ queries in the $i$-th session and $v_{i_t}$ denotes the embedding of the $t$-th query of the $i$-th session. The relationship between these embeddings can be formulated using the three mapping functions above: $\mathbb{F}$ maps a query $q_{i_j}$ to its embedding $v_{i_j}$ in vector space; $\mathbb{G}$ maps a sequence of query embeddings $[v_{i_1}, ..., v_{i_t}]$ to a contextual embedding $c_{i_t}$; and $\mathbb{H}$ maps the contextual embedding to a program $p_i$:

$$v_{i_t} \sim \mathbb{F}(q_{i_t}; \theta_{\mathbb{F}}), \quad c_{i_t} \sim \mathbb{G}(v_{i_1}, ..., v_{i_t}; \theta_{\mathbb{G}}), \quad p_i \sim \mathbb{H}(c_{i_t}; \theta_{\mathbb{H}})$$
$$1 \le t \le |s_i|$$

By assuming that each query is embedded independently, we can reduce the last term in Equation (2) as follows:

$$P(p_i | q_{i_1}, ..., q_{i_t}) = P(p_i | c_{i_t}) \cdot P(c_{i_t} | v_{i_1}, ..., v_{i_t}) \cdot \prod_{j=1}^{t} P(v_{i_j} | q_{i_j})$$

We model the query embedding function $\mathbb{F}(\cdot)$ and the contextual function $\mathbb{G}(\cdot)$ by organizing two Long Short-Term Memory (LSTM) [12] models in a hierarchical manner. The decision function $\mathbb{H}(\cdot)$ uses a feedforward neural network. Before we detail our model architecture, we provide a brief overview of LSTMs.

**Long Short-Term Memory Networks.** LSTM [12] networks are well-known for being able to capture long-range contextual dependencies over input sequences. This is accomplished using a sequence of memory cells to store and memorize historical information, where each memory cell contains three gates (input gate, forget gate, and output gate) to control the information flow. The gating mechanism enables the LSTM to handle the gradient vanishing/explosion problem for long sequences of inputs.

Given an input sequence $\mathbf{x} = (x_1, ..., x_T)$, an LSTM model outputs a sequence of hidden vectors $\mathbf{h} = (h_1, ..., h_T)$. A memory cell at position $t$ digests the input element $x_t$ and previous state information $h_{t-1}$ to produce updated state $h_t$ using the standard LSTM equations [12]. In this paper, we refer to the size of the output vector $h$ as the LSTM size.

## 3.2 Query Representation

Since query strings serve as the sole input in our model, an expressive query representation is essential to accurate predictions. We represent each query as a sequence of elements (words or characters); each element is passed through a lookup layer and projected into a $d$-dimensional vector, thereby representing the query as an $m \times d$ matrix ($m$ is the number of elements in the query). We consider three variations of this representation:

**(1) Character-level** representation, which encodes a query as a sequence of characters and the lookup layer converts each character to a one-hot vector. In this case, $m$ would be the number of characters in the query and $d$ would be the size of the character dictionary of the entire dataset.

**(2) Word-level** representation, which encodes the query as a sequence of words, and the word vectors are read from a pre-trained word embedding, e.g., word2vec [19]. In this case, $d$ would be the dimensionality of the word embedding.

**(3) Combined** representation, which combines both the character-level and word-level representations by feeding the representations to two separate query embedding functions $\mathbb{F}_c$ and $\mathbb{F}_w$, respectively, and then concatenating the two learned vectors $v_c$ and $v_w$ as the combined query embedding vector.

Our intuition for these different representations is as follows: Based on our query log analysis, we observe many unsatisfactory responses due to ASR errors. For example, voice queries intended for the program "Caillou" (a Canadian children's education television series) are often recognized as "Cacio" or "you". Capturing such variations with a word-level representation would likely suffer from data sparsity issues. On the other hand, initializing a query through word embedding vectors would encode words in a semantic vector

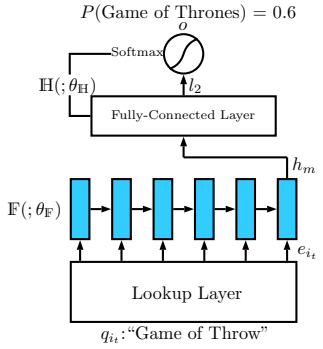



**Figure 2: Architecture of the Basic Model.**

space, which would help in matching queries to programs based on semantic relatedness (e.g., the query "Portland Trail Blazers" is semantically similar to the intended program "NBA basketball" without any words in common). Word embeddings are also useful for recognizing semantically-similar contextual clues such as "Search", "Find" or "Watch". With a character-level representation, such similarities would need to be learned from scratch. Whether the benefits of either representation balance its drawbacks is an empirical question we study through experiments, but we expect that a combined representation can capture the best of both worlds.

### 3.3 Basic Model

In the basic context-independent model, queries in a session are assumed to be independent and thus we do not attempt to model context. That is, each query is treated as a complete sample for model inference and prediction. The mapping function $\Theta$ from query to program in Equation (1) can be simplified as follows:

$$\begin{aligned}\Theta &\sim \arg\max_{\theta} \prod_{i=1}^{|D|}\prod_{t=1}^{|s_i|} P(p_i|q_{i_t}) \\ &= \arg\max_{\theta} \prod_{i=1}^{|D|}\prod_{t=1}^{|s_i|} P(p_i|v_{i_t})P(v_{i_t}|q_{i_t})\end{aligned} \tag{3}$$

$$v_{i_t} \sim \mathbb{F}(q_{i_t};\theta_{\mathbb{F}}), \quad p_i \sim \mathbb{H}(v_{i_t};\theta_{\mathbb{H}}), \quad 1 \le t \le |s_i|$$

Here, the program $p_i$ is only dependent on the current query $q_{i_t}$. The contextual function $\mathbb{G}(\cdot)$ is modeled as an identity function since there is no context from our assumption.

The architecture of the basic model is shown in Figure 2. In the bottom, we use an LSTM as our query embedding function $\mathbb{F}(\cdot)$. The text query is projected into an $m \times d$ dimensional matrix through the lookup layer, then fed to the LSTM, which has $m$ memory cells and each cell processes an element vector. The hidden state at the last time step $h_m$ is used as the query embedding vector $v$. At the top, there is a fully-connected layer followed by a softmax layer for learning the classification function $\mathbb{H}(\cdot)$. The fully-connected layer consists of two linear layers with one element-wise activation layer in between. Given the query embedding vector $v$ as input, the fully-connected layer computes the following:

$$l_2 = W_{h_2} \cdot \sigma(W_{h_1} \cdot v + b_{h_1}) + b_{h_2}$$

where the $W$ terms are the weight matrices and the $b$ terms are bias vectors. We use the tanh function as the non-linear activation

**Algorithm 1** Training the Basic Model

```
1:  for each session s_i in the dataset i = 1...|D| do
2:      for each query q_{i_t} in session s_i with t = 1...|s_i| do
3:                                              ▷ Forward Prediction Start
4:          e_{i_t} = encode(q_{i_t})
5:          h_{1,...,m} = LSTM:forward(e_{i_t})
6:          l_2 = FC:forward(h_m)
7:          o = softmax:forward(l_2)
8:          loss = criterion:forward(o, p_i)
9:                                          ▷ Backward Propagation Start
10:         grad_criterion = criterion:backward(o, p_i)
11:         grad_soft = softmax:backward(l_2, grad_criterion)
12:         grad_linear = FC:backward(h_m, grad_soft)
13:         grad_lstm = zeros(m, lstm_size)
14:         grad_lstm[m] = grad_linear
15:         LSTM:backward(e_it, grad_lstm)
16:         update_parameters()
```

function $\sigma$, which is commonly adopted in neural network architectures. The softmax layer normalizes the vector $l_2$ to an $L1$ norm vector $o$, with each output score $o[p_j]$ denoting the probability of producing program $p_j$ as output:

$$o[p_j] = \frac{\exp(l_2[p_j] - \text{shift})}{\sum_{p_k=1}^{|\Phi|}\exp(l_2[p_k] - \text{shift})}$$

where $\text{shift} = \max_{p_k=1}^{|\Phi|} l_2[p_k]$, $|\Phi|$ is the total number of programs in the dataset. We adopt the negative log-likelihood loss function to train the model, which is derived from Equation (3):

$$\begin{aligned}L &= -\sum_{i=1}^{|D|}\sum_{t=1}^{|s_i|}\log P(p_i|q_{i_t}) + \lambda \cdot \|\langle\theta_{\mathbb{F}},\theta_{\mathbb{G}},\theta_{\mathbb{H}}\rangle\|^2 \\ &= -\sum_{i=1}^{|D|}\sum_{t=1}^{|s_i|}\log o_{i_t}[p_i] + \lambda \cdot \|\langle\theta_{\mathbb{F}},\theta_{\mathbb{G}},\theta_{\mathbb{H}}\rangle\|^2\end{aligned}$$

where $o_{i_t}$ is the score vector computed from query $q_{i_t}$ and $p_i$ is the true program for session $i$; $\lambda$ is the regularization weight and $\langle\theta_{\mathbb{F}},\theta_{\mathbb{G}},\theta_{\mathbb{H}}\rangle$ is the set of model parameters. The optimization goal is to minimize the loss criterion $L$.

The training process is shown in Algorithm 1. The overall structure is to iterate over each query in all sessions to perform the forward prediction and backward propagation operations. The *forward* phase follows our model architecture in Figure 2. A query is first encoded as a matrix in Line 4 by specifying the encoding method (i.e., character, word, or combined). Lines 5 and 6 feed the input matrix to the LSTM and fully-connected (FC) layer sequentially. In the *backward* phase, each module requires the original inputs and the gradients propagated from its upper layer to compute the gradients with respect to the inputs and its own parameters. It is worth noting that in Lines 13-15 the gradients grad_lstm are initialized to zero for the first $m-1$ cells. This is because in the forward phase, we only use the last LSTM state $h_m$ as the query embedding vector for the upper layers. Line 16 performs gradient descent to update model parameters. All the forward and backward functions used here are written as black box operations, and we refer interested readers elsewhere [7] for more details.

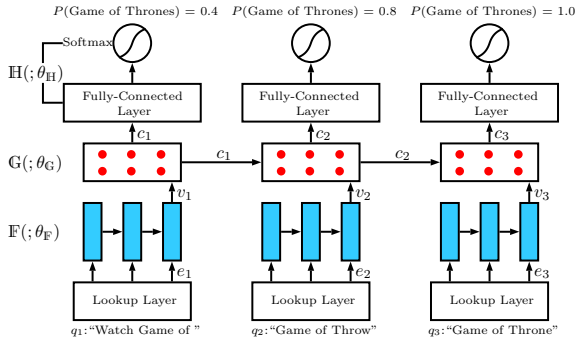



**Figure 3: Architecture of the Full Context Model.**

## 3.4 Full Context Model

We propose two approaches to model context: the full context model (presented here) and the constrained context model (presented next). The architecture of the full context model is shown in Figure 3, which uses the basic model as a building block. We use another LSTM (the dotted rectangle in the middle of Figure 3) to learn the contextual function $\mathbb{G}(v_1, ..., v_t; \theta_{\mathbb{G}})$. Previous query embedding vectors $[v_1, ..., v_{t-1}]$ are encoded as a context vector $c_{t-1}$, which is combined with the current query embedding vector $v_t$ and fed to the LSTM memory cell at time $t$. This allows the LSTM to find an optimal combination of signals from prior context and the current query. For sessions with a single underlying intent (i.e., the user is consistently looking for a specific program), the model can learn the relatedness between successive queries and continuously reinforce confidence in the true intent. In reality, context is sometimes irrelevant, which might introduce noise. When the context diverges too much from the current query embedding, the model should be able to ignore the noisy context signals to reduce their negative impact.

We adopt a many-to-many hierarchical architecture. The query embedding layer $\mathbb{F}(\cdot)$ and the classification layer $\mathbb{H}(\cdot)$ are applied to each query for program prediction at each time step $t$. We hope to find the true user intent as early as possible to reduce interactions between the user and our product. The parameters of the query embedding layer $\mathbb{F}(\cdot)$, as well as the classification layer $\mathbb{H}(\cdot)$, are shared by all queries regardless of their position in the session. For instance, two identical queries with different positions in a session will have the same query embedding vector. Except for the contextual layer $\mathbb{G}(\cdot)$, all other modules (e.g., query embedding, fully-connected layer, softmax layer, loss function) remain the same as in the basic (context-independent) model.

The training process for this model (Algorithm 2) starts with forward predictions for multiple queries in the session (Lines 2-6). Similar to Algorithm 1, only the last LSTM state $h_m$ is selected as the query embedding vector (Line 6). Since sessions can have a variable number of queries, we dynamically clone a new LSTM to ingest the query at time $t$ (i.e., LSTM[t] in Line 5) when the arriving query results in a longer session than all previously seen sessions. Line 7 utilizes another LSTM model to compute the context from sequential query embeddings. Lines 8-16 perform forward predictions and backward propagations for multiple queries in the classification layer. The queries are processed in a sequential manner such that

**Algorithm 2** Training the Full Context Model

1: **for** each session $s_i$ in the dataset $i = 1...|D|$ **do**
2: $v$ = zeros($|s_i|$, lstm_size) ▷ query embedding vectors
3: **for** each query $q_{i_t}$ with $t = 1...|s_i|$ **do**
4: $e_{i_t}$ = encode($q_{i_t}$)
5: $h_{1,...,m}$ = LSTM[t]:forward($e_{i_t}$)
6: $v_t = h_m$
7: $c_{1,...,|s_i|}$ = C_LSTM:forward($v_{1,...,|s_i|}$) ▷ contextual vectors
8: grad_linear = zeros($|s_i|$, lstm_size)
9: **for** each query $q_{i_t}$ with $t = 1...|s_i|$ **do**
10: $l_2$ = FC:forward($c_t$)
11: $o$ = softmax:forward($l_2$)
12: loss = criterion:forward($o$, $p_i$)
13: session_loss = session_loss + $loss$
14: grad_criterion = criterion:backward($o$, $p_i$)
15: grad_soft = softmax:backward($l_2$, grad_criterion)
16: grad_linear[$t$] = FC:backward($c_t$, grad_soft)
17: grad_context = C_LSTM:backward($v_{1,...,|s_i|}$, grad_linear)
18: **for** each query $q_{i_t}$ with $t = 1...|s_i|$ **do**
19: grad_lstm = zeros($m$, lstm_size)
20: grad_lstm[$m$] = grad_context[$t$]
21: LSTM[t]:backward($e_{i_t}$, grad_lstm)
22: update_parameters()

for each query all forward operations are immediately followed by all backward operations before moving to the next query. Lines 17-21 propagate the gradients through the contextual and embedding LSTMs. Line 22 updates model parameters for each session by optimizing the session loss in Line 13.

It is important to note that the prediction task is applied at the query level: our model tries to predict the program after *each* query in the session. The alternative is to optimize for program prediction given *all* queries in the session—this is a much easier task, since the entire session has been observed. It also defeats the purpose of our setup since we wish to satisfy viewer intents as quickly as possible.

## 3.5 Constrained Context Model

Finally, we explore a variant that we call the *constrained* context model. The model architecture is the same as the full context model (Figure 3). The difference, however, lies in how we learn the model. For the constrained context model, we adopt a pre-training strategy as follows: we first train the basic model (Algorithm 1) and then use the learned LSTM parameters to initialize the constrained context model's query embedding layer. The embedding layer is then fixed and purely used for generating query embeddings. That is, Lines 18-21 are removed from Algorithm 2.

Our intuition behind this model is to restrict the search space during model training, aiming to reduce the complexity of optimization compared to the full context model. Whether this reduction in optimization complexity is beneficial to the prediction task is an empirical question we study in the following sections.

# 4 EXPERIMENTAL SETUP

## 4.1 Data Preparation

We collected voice queries submitted to Xfinity X1 remote controllers during the week of Feb. 22 to 28, 2016, a total of 32.3M queries from 2.5M unique viewers. Based on preliminary analyses,

we selected 45 seconds as the threshold for dividing successive queries into sessions, yielding 20.0M sessions.

To build a training set for supervised learning, we need the true user intent for each session. We automatically extracted noisy labels by examining what the viewer watched after the voice session; this exactly parallels inferring user intent from clickthrough data in the web domain. If the viewer began watching a program $p$ at most $K$ seconds after the last query in the voice session $v$ and kept watching it for at least $L$ seconds, we label the session with $p$. The selection of $K$ and $L$ represents a balance between the quantity and quality of collected labels. After some initial exploration, we set $K$ to 30 seconds and $L$ to 150 seconds, which yields a good balance between data quantity and quality (based on manual spot-checking). Using these parameters, we extracted 13.0M session-program pairs. These sessions contain mixed modalities, as in reality users navigate with a combination of voice queries and keypad entry.

Without any further processing, these voice queries might reflect arbitrary intent (e.g., "closed caption on", "the square root of eighty one", or "change to channel 36"). In order to limit ourselves to voice sessions with a single clear intent, we used two heuristics as follows: First, we define a way to reliably predict whether a query is program-related (i.e., the query is primarily associated with a TV series, movie, video, or sports program). We obtain this from the deployed X1 platform, which categorizes every query into one of many action types based on existing hand-crafted patterns and partial string matching. A query is program-related if it is categorized as one of the following: {SERIES, MOVIE, MUSICVIDEO, SPORTS}. Based on this knowledge, we restricted our data to sessions in which over 2/3 of queries are program-related and the final query in the session is also program-related. Since channel changes are a large portion of the data, this reduces the number of labeled pairs to 2.1M.[1]

Second, we computed the normalized edit distance between each query pair in the session, and kept only those sessions where *any* pair of queries has a distance less than 0.5. Our goal here is to ensure that there is at least *some* cohesion in the sessions. This heuristic has a relatively minor effect: the resulting filtered dataset contains 1.96M sessions in total.

Naturally, data pre-processing plays an important role in any problem setup, and one might wonder if our heuristics might generate a dataset that favors our approach. This is unlikely because the data filtering is a function of two independent decisions: (i) the action types are chosen by the currently-deployed system, and (ii) viewing behaviors after the voice sessions are purely dependent on users. Both decisions are unrelated to our methods. Furthermore, as we are ultimately interested in improving our production system, it is only natural to bootstrap off existing log data, much like the development of web search.

From this data, we sampled five splits: a training set used in all experiments, and two groups of development and test sets. The first development and test sets contain only single-query sessions, called SingleDev and SingleTest. These are used to study whether the context-based models hurt accuracy in sessions without context. The second group contains only multiple-query sessions (i.e., at least two queries in each session), called MultiDev and MultiTest.

[1] Although channel changes can be handled like programs, we decided to ignore them due to the additional complexity of regional variations (e.g., HBO West and HBO East are two different channels and cannot be disambiguated solely based on the query).

| Dataset | sessions | queries | avg. session len | avg. query len |
|---|---|---|---|---|
| Train | 126016 | 181058 | 1.44 | 2.34 |
| SingleDev | 24792 | 24792 | 1.00 | 2.40 |
| SingleTest | 24572 | 24572 | 1.00 | 2.36 |
| MultiDev | 28427 | 82828 | 2.91 | 2.30 |
| MultiTest | 28173 | 82272 | 2.92 | 2.30 |

**Table 1: Dataset statistics.**

To build the global set of programs $\Phi$, we only kept a program if there are at least 50 associated sessions in the training set, yielding 471 programs that account for 77.8% of all the viewing behaviors of the 1.96M filtered sessions during that week. Statistics for each of the splits after all pre-processing are summarized in Table 1.

## 4.2 Model Training

In total, we have three options for query representation, *char*, *word*, and *combined* (Section 3.2), and three options for the model, *basic*, *full context*, and *constrained context* (Sections 3.3-3.5). Therefore, we have a total of nine experimental settings, by crossing the three representations with the three models.

The entire dataset contains 80 distinct characters in total, which means that the size of the one-hot vector used in the *char* setting is 80. For the word representation, we used 300-dimensional GloVe word embeddings [22] to encode each word, which is trained on 840 billion tokens and freely available. The word vocabulary of our dataset is 20.4K, with 1759 words not found in the GloVe word embeddings. Unknown words were randomly initialized with values uniformly sampled from [-0.05, 0.05].

During training, we used the stochastic gradient descent algorithm together with RMS-PROP [32] to iteratively update the model parameters. The learning rate was initially set to $10^{-3}$ and then decreased by a factor of three when the development set loss stopped decreasing for three epochs. The maximum number of training epochs was 50. For the constrained context model, the number of pre-training epochs was selected as 15. The output size of the LSTMs was set to 200 and the size of the linear layer was set to 150. The regularization weight $\lambda$ was chosen as $10^{-4}$. At test time, we selected for evaluation the model that obtained the highest P@1 accuracy on the development set. Our models were implemented using the Torch framework. We ran all experiments on a server with two 8-core processors (Intel Xeon E5-2640 v3 2.6GHz) and 1TB RAM, with each experiment running on 6 CPU threads.

## 4.3 Baselines

We compared our methods against the following baselines:

**TF-IDF/BM25**: We built a 3-gram (character-level) inverted index of the program set $\Phi$. During retrieval, the matching score is computed on 3-gram overlaps between query and programs using TF-IDF or Okapi BM25 weighting ($k_1 = 1.2$, $b = 0.75$).

**Learning-to-rank** $\text{SVM}^{\text{rank}}$ [15]: We first used BM25 to retrieve at most 20 candidate programs per query, then designed three types of features for each pair of query and candidate program: (1) BM25 score, (2) max/mean/min value of cosine similarities between word embeddings of the query and the candidate program, and (3) popularity score of the candidate program.

| ID | Model | Query | P@1 | P@5 | MRR |
|---|---|---|---|---|---|
| 1 | TF-IDF | 3-gram | $0.895^{10}$ | $0.937^{10}$ | $0.923^{10}$ |
| 2 | BM25 | 3-gram | $0.907^{1,10}$ | $0.939^{10}$ | $0.927^{10}$ |
| 3 | $SVM^{rank}$ | - | $0.917^{1,2,10}$ | $0.951^{1,2,10}$ | $0.933^{1,9}$ |
| 4 | DSSM | 3-gram | $0.918^{1,2,10}$ | $0.949^{1,10}$ | $0.931^{1,10}$ |
| 5 | DSSM+S | 3-gram | $0.916^{1,10}$ | $0.949^{1,10}$ | $0.936^{1,10}$ |
| 6 | Basic | char | $0.944^{1\text{-}5,9,10}$ | $0.953^{1,2,9,10}$ | $0.962^{1\text{-}5,10}$ |
| 7 | Basic | word | $0.943^{1\text{-}5,9,10}$ | $0.953^{1,2,9,10}$ | $0.962^{1\text{-}5,10}$ |
| 8 | Basic | comb | $\mathbf{0.947}^{1\text{-}5,9,10}$ | $\mathbf{0.955}^{1,2,9,10}$ | $0.964^{1\text{-}5,10}$ |
| 9 | Basic+S | comb | $0.923^{1,2,10}$ | $0.938^{10}$ | $0.953^{1\text{-}5,10}$ |
| 10 | Context-f | char | 0.753 | 0.794 | 0.837 |
| 11 | Context-f | word | $0.926^{1,2,10}$ | $0.942^{10}$ | $0.959^{1\text{-}5,10}$ |
| 12 | Context-f | comb | $0.932^{1,2,10}$ | $0.947^{10}$ | $\mathbf{0.967}^{1\text{-}5,9}$ |
| 13 | Context-c | char | $0.938^{1\text{-}5,9,10}$ | $0.950^{1,9,10}$ | $0.963^{1\text{-}5,10}$ |
| 14 | Context-c | word | $0.943^{1\text{-}5,9,10}$ | $0.950^{1,9,10}$ | $0.961^{1\text{-}5,10}$ |
| 15 | Context-c | comb | $0.944^{1\text{-}5,9,10}$ | $0.953^{1,2,9,10}$ | $0.963^{1\text{-}5,10}$ |

**(a) single-query**

| ID | Model | Query | P@1 | P@5 | MRR | QR |
|---|---|---|---|---|---|---|
| 1 | TF-IDF | 3-gram | $0.518^{10}$ | $0.593^{10}$ | $0.543^{10}$ | $0.932^{10}$ |
| 2 | BM25 | 3-gram | $0.533^{1,10}$ | $0.596^{10}$ | $0.565^{1,10}$ | $0.947^{1,10}$ |
| 3 | $SVM^{rank}$ | - | $0.547^{1,2,10}$ | $0.621^{1,2,10}$ | $0.582^{1,2,10}$ | $0.962^{1,2,10}$ |
| 4 | DSSM | 3-gram | $0.568^{1,2,3,10}$ | $0.617^{1,2,10}$ | $0.584^{1,2,10}$ | $1.001^{1,2,3,4,10}$ |
| 5 | DSSM+S | 3-gram | $0.550^{1,2,10}$ | $0.618^{1,2,10}$ | $0.576^{1,2,10}$ | $0.972^{1,2,10}$ |
| 6 | Basic | char | $0.605^{1\text{-}5,10}$ | $0.647^{1\text{-}5,10}$ | $0.690^{1\text{-}5,10}$ | $1.108^{1\text{-}5,10}$ |
| 7 | Basic | word | $0.609^{1\text{-}5,10}$ | $0.644^{1\text{-}5,10}$ | $0.677^{1\text{-}5,10}$ | $1.086^{1\text{-}5,10}$ |
| 8 | Basic | comb | $0.614^{1\text{-}5,9,10}$ | $0.651^{1\text{-}5,10}$ | $0.687^{1\text{-}5,10}$ | $1.113^{1\text{-}5,9,10}$ |
| 9 | Basic+S | comb | $0.596^{1\text{-}5,10}$ | $0.645^{1\text{-}5,10}$ | $0.697^{1\text{-}5,10}$ | $1.061^{1\text{-}5,10}$ |
| 10 | Context-f | char | 0.482 | 0.532 | $0.580^{1}$ | 0.856 |
| 11 | Context-f | word | $0.599^{1\text{-}5,10}$ | $0.638^{1\text{-}5,10}$ | $0.687^{1\text{-}5,10}$ | $1.075^{1\text{-}5,10}$ |
| 12 | Context-f | comb | $0.598^{1\text{-}5,10}$ | $0.643^{1\text{-}5,10}$ | $0.688^{1\text{-}5,10}$ | $1.039^{1\text{-}5,10}$ |
| 13 | Context-c | char | $0.639^{1\text{-}12}$ | $0.684^{1\text{-}12}$ | $0.731^{1\text{-}12}$ | $1.117^{1\text{-}7,9\text{-}12}$ |
| 14 | Context-c | word | $0.639^{1\text{-}12}$ | $0.683^{1\text{-}12}$ | $0.729^{1\text{-}12}$ | $1.112^{1\text{-}7,9\text{-}12}$ |
| 15 | Context-c | comb | $\mathbf{0.643}^{1\text{-}12}$ | $\mathbf{0.687}^{1\text{-}12}$ | $\mathbf{0.734}^{1\text{-}12}$ | $\mathbf{1.128}^{1\text{-}12}$ |

**(b) multiple-query**

**Table 2: Model effectiveness on single-query (left) and multiple-query (right) sessions. The second column denotes the model: baselines compared to the basic, full context (Context-f), and constrained context (Context-c) models. The third column indicates the query representation. Remaining columns show evaluation metrics. Superscripts indicate the row indexes for which a metric difference is statistically significant at $p < 0.01$. Rows are numbered in the first column for convenience.**

**DSSM** [13]: This neural ranking model for web search uses word hashing to model interactions between queries and programs at the level of character 3-grams. This method is an appropriate baseline for our problem since it can handle ASR transcription errors, unlike neural ranking models based on word matching [9, 21, 24].

**DSSM+S**: We train and evaluate DSSM by concatenating queries in one session with a special boundary token between neighboring queries. This variant can be considered a context-aware baseline, where we aim to study whether simple concatenation is able to capture context signals in a session.

**Basic**: We applied the basic model described in Section 3.3 with character-level, word-level, and combined representations.

**Basic+S**: Same as above, with a concatenated query representation.

## 5 RESULTS

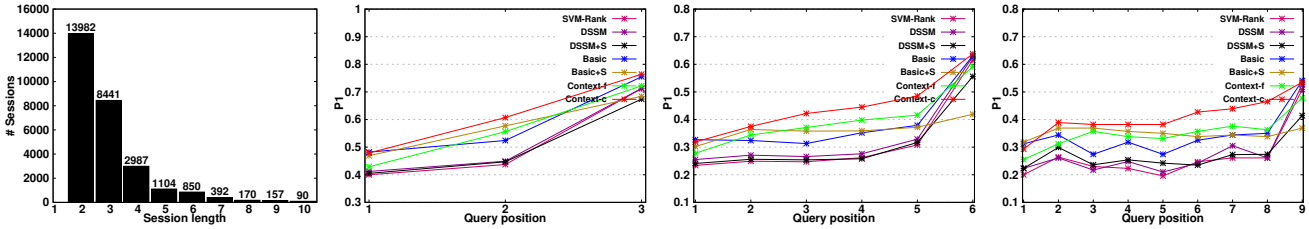
We used four metrics in our evaluation, averaged over all queries: precision at one (P@1), precision at five (P@5), Mean Reciprocal Rank (MRR), and Query Reduction (QR). The first three are standard retrieval metrics and don't require an explanation. QR is a measure of how many queries a viewer has "saved" in a session. For a session with $n$ queries, the number of reductions is $n-i$ if the model returns the correct prediction at the $i$-th query, which means that the viewer does not need to issue the next $n-i$ queries, hence a *reduction* of $n-i$. We average this metric over all sessions.

There are few important questions our experiments were designed to answer: First, how do our proposed models compare against lexical overlap, a learning-to-rank baseline, and an existing neural ranking model? Second, within our model, which is the most effective query representation (character, word, or combined)? Finally, what are the important differences between single query and multiple-query sessions, and which is the most effective context model: ignoring context, query concatenation, full context, or constrained context?

Results for the single-query and multiple-query sessions, on the SingleTest and MultiTest splits, respectively, are shown in Table 2. Each row represents an experimental condition (numbered for convenience). The second column specifies the model: "Context-f" and "Context-c" denote the full and constrained context models, respectively. The third column indicates the query representation (char, word, or combined), and the remaining columns list the various evaluation metrics. Superscripts indicate the row indexes for which a metric difference is statistically significant ($p < 0.01$) based on Fisher's two-sided, paired randomization test [29]. A dash symbol "-" connecting two indices "$a$-$b$" is shorthand for $a, \ldots, b$.

Let's first consider the non-context baselines (numbered 1–4 and 6–8): TF-IDF and BM25 achieve fairly high accuracies (P@1 of 0.895 and 0.907) on single-query sessions. The effectiveness of these two simple baselines is not surprising, since single-query sessions are by construction the "easy queries" in which users reach their intended program in only a single voice query (but may include keypad navigation). The $SVM^{rank}$ predictor achieves better accuracy than the BM25 baseline because it also takes advantage of word embeddings to consider semantic relatedness as well as the popularity priors of programs in a supervised setting. DSSM achieves comparable performance to the $SVM^{rank}$ approach, significantly outperforming both TF-IDF and BM25. We also noticed that our basic (context-independent) model outperforms DSSM by quite a bit. We identified two main reasons: (1) There are about 2% sports-related queries in which the user searched for a particular sports team (e.g., "Detroit Red Wings") but ended up watching a program (e.g., "NHL Hockey") that shares no common 3-grams with the query; (2) Major ASR errors sometimes results in very little lexical overlap between the query and the intended program title (e.g., "Dr. Seuss's The Lorax" is transcribed as "The Laura").

Turning our attention to non-context models on the multiple-query sessions in Table 2(b), we see that accuracy drops significantly




**(a) Session Length Distribution** **(b) Short Sessions** **(c) Medium Sessions** **(d) Long Sessions**

**Figure 4: Context analysis of the multiple-query session (MultiTest) test set. The leftmost plot shows the distribution of session lengths. Subfigures (b)-(d) show the average P@1 score at different positions (i.e., the $i$-th query) in the session.**

compared to single-query sessions. The relative effectiveness of the various methods is consistent, but we observe a much wider range of prediction accuracy separating the simple methods (TF-IDF and BM25) from the more sophisticated models (DSSM and the basic model). These results suggest that neural network models are able to better capture signals beyond lexical overlap and the few other manually-defined features we employed.

For the first question, we observe that in the basic and constrained context models, the word-level query representation is quite close to the character-level query representation. However, in nearly all conditions, across nearly all metrics, the combined condition further improves (albeit only slightly) upon both representations, which shows that character-level and word-level representations provide signals that supplement each other.

In terms of the context models, the constrained context model significantly outperforms all others, including the basic, query concatenation, and full context models. We observe that query concatenation hurts the effectiveness of the DSSM and basic models. From the output logs, we find that query concatenation is less robust to noise in the session data, both at training and at inference time. For example, consider sessions beginning with a general-purpose query such as "Find me all free movies on demand": the context of all subsequent queries will be "polluted" by irrelevant information, from which the model might not be able to recover.

In contrast, modeling query dependencies through an LSTM model is more effective. Since the context models copy their query embedding layers from the basic models, we conclude that the upper LSTM layer is able to capture contextual information to improve prediction. Note that the full and constrained models share exactly the same architecture; the only difference lies in whether or not we back-propagate to the query embedding layer during training—the constrained model was designed to restrict the model search space. The effectiveness gap on the multiple-query session dataset (0.599 vs 0.643) demonstrates that the query embedding layer obtained by the constrained context model through pre-training is of higher quality. This is likely due to insufficient data for the full context model to effectively learn parameters for both LSTM levels. However, a caveat: it is conceivable that with even more training data, the full context model will improve. But as it currently stands, the constrained context model displays a better ability to exploit contextual information for predicting viewers' intent. Overall, for the multiple-query sessions, the constrained model with the combined representation yields a 21% relative improvement over BM25 and 13% over DSSM in terms of P@1.

For single-query sessions, we want to make sure that the context models do not "screw up" these queries. We confirm that this is indeed the case. It is no surprise that the basic model performs the best on single-query sessions: since there is no context to begin with, all the "contextual machinery" of the richer models can only serve as a distraction. We find that the constrained model with the combined representation (best condition above) still performs well—slightly worse than the basic model with the combined representation, although the differences are not statistically significant. It is also interesting to note that the full context model with the character representation is terrible, suggesting that the search space is too large given the combination of longer query representations and session-level optimization.

## 5.1 Context Analysis

To better understand how our models take advantage of context, we focused on multiple-query sessions and examined how accuracy evolves during the course of a session. Results are shown in Figure 4. The leftmost plot shows the histogram of session lengths (i.e., number of queries in each session) in the MultiTest split; each bar is annotated with the actual count. In Figures 4(b)-(d), we show the average P@1 score from MultiTest at different positions in the session (on the $x$ axis), i.e., at the first query in the session, the second query, etc. For illustrative purposes we focus on "short" sessions with a length of three (8441 sessions), "medium" sessions with a length of six (850 sessions), and "long" sessions with a length of nine (157 sessions). For clarity, in all cases the models used the combined query representation.

We observe several interesting patterns in Figures 4(b)-(d). First, for the non-context models ($\text{SVM}^{\text{rank}}$, DSSM, and basic), the accuracy stays flat until the final query (except small fluctuations due to noise). Accuracy for the final query rises significantly because the viewer finally found what she was looking for (and thus is likely to be an "easy" query). Also, we can see that DSSM with the concatenated query representation (DSSM+S) is not able to capture context clues in the sessions, illustrated by a curve as flat as the non-context models. The Basic+S model is interesting: At the beginning of sessions, it is able to outperform the basic model, yet its effectiveness degrades as the session progresses. At the end of sessions, it is consistently worse than the basic model. This suggests that simple concatenation is prone to accumulating too much noise during longer sessions, eventually outweighing the benefits of having context. However, for the context-aware models (Context-f and Context-c), we observe a consistent increase in the accuracy

| **Session** | | Cacio : You : You : Caillou | Sienna cover : Color : Casey undercover |
|---|---|---|---|
| **Program** | | Caillou | K.C. Undercover |
| **Model** | **Query** | **Example 1** | **Example 2** |
| BM25 | - | Pacific Rim : Now You See Me : Now You See Me : ★ | Recovery Road : College Basketball : ★ |
| $SVM^{rank}$ | - | Pacific Rim : Now You See Me : Now You See Me : ★ | Recovery Road : Dora the Explorer : ★ |
| DSSM | 3-gram | Pacific Rim : Young : Young : ★ | ★ : College Basketball : ★ |
| DSSM+S | 3-gram | Pacific Rim : The Young and the Restless : The Young and the Restless : ★ | ★ : College Basketball : ★ |
| Basic | char | ★ (0.81) : ★ (0.80) : ★ (0.80) : ★ (1.0) | ★ (0.76) : Carolina (0.07) : ★ (0.99) |
| Basic | word | Child Genius (0.03) : ★ (0.57) : ★ (0.57) : ★ (1.0) | Recovery Road (0.48) : ★ (0.08) : ★ (0.75) |
| Basic | comb | Paw Patrol (0.17) : ★ (0.83) : ★ (0.83) : ★ (1.0) | ★ (0.37) : Magic Mike XXL (0.31) : ★ (0.98) |
| Basic+S | comb | Paw Patrol (0.15) : Paw Patrol (0.25) : ★ (0.75) : ★ (0.98) | ★ (0.34) : ★ (0.20) : ★ (0.85) |
| Context-f | char | Lego Ninjago (0.30) : ★ (0.79) : ★ (0.90) : ★ (0.99) | ★ (0.43) : ★ (0.67) : ★ (0.89) |
| Context-f | word | Paw Patrol (0.30) : ★ (0.62) : ★ (0.98) : ★ (1.0) | ★ (0.29) : ★ (0.65) : ★ (1.0) |
| Context-f | comb | Lego Ninjago (0.03) : ★ (0.60) : ★ (0.98) : ★ (1.0) | ★ (0.41) : ★ (0.54) : ★ (0.99) |
| Context-c | char | ★ (0.96) : ★ (0.99) : ★ (0.99) : ★ (1.0) | ★ (0.81) : ★ (0.96) : ★ (0.99) |
| Context-c | word | Wallykazam (0.07) : ★ (0.59) : ★ (0.86) : ★ (1.0) | ★ (0.89) : ★ (0.80) : ★ (1.0) |
| Context-c | comb | Paw Patrol (0.17) : ★ (0.93) : ★ (1.0) : ★ (1.0) | ★ (0.65) : ★ (0.83) : ★ (0.97) |

**Table 3: Two sample sessions and top predictions for each model. Each query and prediction in the session is separated by a colon. For each prediction from our models, we show the confidence score. ★ indicates that the model response was correct.**

curves as the session progresses. This demonstrates that the LSTM model is able to better capture context signals in the sessions, and as the model accumulates more context, it can better identify the user's true intent. The constrained context model performs about the same as the basic model at the first query (where there is no context) and the final query (which is easier since the viewer found the desired program). However, the constrained context model outperforms the basic model in the middle of sessions, highlighting the ability of the LSTM to capture context.

In Table 3, we provide two real example sessions to illustrate how each model responds to the sequence of viewer queries. The session is shown in the first row, where each query is separated by a colon. The second row shows the viewer's intent (i.e., ground truth label). The remaining rows show the output of each model; due to space limitations, we only show the top predictions along with their confidence scores for our models. Each prediction in the sequence is also separated by a colon. To save space, we use the symbol ★ to indicate that the prediction is correct.

In the first example (left), the viewer is consistently looking for the program "Caillou", but the production system fails three times in a row due to ASR errors. For the first three queries, all models based on n-gram matching fail (BM25, $SVM^{rank}$, DSSM, DSSM+S) because there is no 3-gram overlap between the queries and the intended program. For our models, both the basic/char and Context-c/char models can predict the correct program from the query "Cacio" with high confidence. However, models with word-level representations all fail for this query. This is no surprise as the word "Cacio" is a rare mis-transcription of the word "Caillou" and thus rarely seen in the training set. For the next two successive queries "You", the basic models are able to succeed with high confidence scores. However, the Basic+S model remains mistaken after the first occurrence of "You", suggesting that concatenation with the previous query "Cacio" serves as a distractor. For both the full and constrained context models, confidence for the second query "You" is higher (thanks to context). This is an example of how contextual clues help, confirming our intuition. Since the second example behaves similarly, we omit a description for space consideration.

## 5.2 Efficiency Analysis

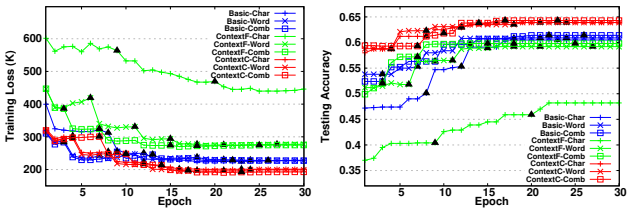
Our final set of experiments examined efficiency in terms of training time and prediction latency, both important consideration for production deployments. Results are shown in Table 4. For each setting in the first two columns, "#Params" shows the total number of parameters in the model, "Training" denotes the training time for each epoch, and "Test" shows the prediction latency per query. "Avg." and "Conf." indicate the average value and the 95% confidence interval of training/test times over 30 epochs. Overall, the training time of all models is less than around 100 minutes per epoch, and the per-query prediction latency is less than 8 milliseconds. Most model configurations converge in the first 20 epochs.

Comparing different query representations, we observe that *combined* has the most number of parameters and was also the slowest to train and test; this is no surprise. For the character-level representation, the size of the one-hot vectors is smaller, resulting in fewer parameters at the query embedding layer. However, the number of characters in a query is much larger, leading to longer training times. With the same query representation, the full context models have more parameters and take longer to train than the corresponding basic and constrained context models, as expected. The constrained context models have the same number of parameters and similar prediction latencies as the full context models since they share the same architecture. However, the constrained context models are much faster to train, suggesting that most of the training effort is spent on the query embedding layer in the full context models.

We plot training loss and testing accuracy curves in Figure 5: (a) shows the training loss curve as a function of epoch, (b) shows the P@1 curve in the MultiTest set at each epoch. The symbol ▲ denotes the epochs where the learning rate was divided by three because development loss had not decreased for three epochs. We see that most models converged within 20 epochs. In the basic

| Model | Query | #Params | Training (min) Avg. | Conf. | Test (ms) Avg. | Conf. |
|---|---|---|---|---|---|---|
| Basic | char | 326,871 | 62.1 | [59.7, 64.7] | 6.4 | [6.2, 6.6] |
| Basic | word | 502,871 | 32.6 | [32.2, 33.0] | 3.0 | [3.0, 3.1] |
| Basic | comb | 758,471 | 94.5 | [91.4, 97.2] | 6.9 | [6.6, 7.0] |
| Context-f | char | 648,471 | 72.1 | [68.4, 74.5] | 6.6 | [6.4, 7.0] |
| Context-f | word | 824,471 | 58.8 | [57.2, 60.8] | 4.0 | [4.0, 4.1] |
| Context-f | comb | 1,210,071 | 102.4 | [100.8, 103.8] | 7.0 | [6.8, 7.2] |
| Context-c | char | 648,471 | 32.1 | [30.9, 33.7] | 6.6 | [6.4, 6.8] |
| Context-c | word | 824,471 | 30.1 | [29.5, 31.2] | 4.0 | [3.9, 4.1] |
| Context-c | comb | 1,210,071 | 42.5 | [41.8, 43.1] | 6.9 | [6.8, 7.0] |

**Table 4: Model efficiency: Column "Training" shows the training time per epoch and "Test" shows prediction latency per query. "Avg." indicates the average value of training/test times, and "Conf." indicates the 95% confidence interval.**




**(a) Training Loss** **(b) Testing Accuracy**

**Figure 5: Training loss and testing accuracy for each epoch; ▲ denotes epochs where the learning rate was reduced.**

and full context models, the *char* representation took longer to converge than *word* or *combined*, which shows that a character-level representation is more difficult to learn. It is also interesting that the gap in training loss between the basic and full context models is larger than the gap in test accuracy, which means that although the full context model is difficult to train, the benefit of context enables it to generalize well. The constrained context model walks a middle ground in terms of model complexity and the ability to capture context information, leading to both lower training loss and higher test accuracy.

## 6 CONCLUSION

Our vision is that future entertainment systems should behave like intelligent agents and support voice interactions. As a first step, we tackle a specific problem, voice navigational queries, to help users find the TV programs they are looking for. We articulate the challenges associated with this task, which we tackle with two ideas: by combining word- and character-level representations of queries, and by modeling session context, both using hierarchically-arranged neural network modules. Results on a large real-world voice query log show that our methods can effectively cope with ambiguity and compensate for ASR errors. Indeed, we allow viewers to talk to their TVs, and for customers who learn of this feature for the first time, it is a delightful experience!

## REFERENCES


[1] Alex Acero, Neal Bernstein, Rob Chambers, Yun-Cheng Ju, Xinggang Li, Julian Odell, Patrick Nguyen, Oliver Scholz, and Geoffrey Zweig. 2008. Live Search for Mobile: Web Services by Voice on the Cellphone. In *ICASSP*.

[2] Paul N. Bennett, Ryen W. White, Wei Chu, Susan T. Dumais, Peter Bailey, Fedor Borisyuk, and Xiaoyuan Cui. 2012. Modeling the Impact of Short- and Long-term Behavior on Search Personalization. In *SIGIR*.

[3] Huanhuan Cao, Derek Hao Hu, Dou Shen, Daxin Jiang, Jian-Tao Sun, Enhong Chen, and Qiang Yang. 2009. Context-Aware Query Classification. In *SIGIR*.

[4] Ciprian Chelba and Johan Schalkwyk. 2013. Empirical Exploration of Language Modeling for the google.com Query Stream as Applied to Mobile Voice Search. In *Mobile Speech and Advanced Natural Language Solutions*.

[5] Fabio Crestani and Heather Du. 2006. Written Versus Spoken Queries: A Qualitative and Quantitative Comparative Analysis. *JASIST* 57, 7 (2006), 881–890.

[6] Junlan Feng and Srinivas Bangalore. 2009. Effects of Word Confusion Networks on Voice Search. In *EACL*.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[8] Dongyi Guan, Sicong Zhang, and Hui Yang. 2013. Utilizing Query Change for Session Search. In *SIGIR*.

[9] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. 2016. A Deep Relevance Matching Model for Ad-hoc Retrieval. In *CIKM*.

[10] Ido Guy. 2016. Searching by Talking: Analysis of Voice Queries on Mobile Web Search. In *SIGIR*. 35–44.

[11] Ahmed Hassan Awadallah, Ranjitha Gurunath Kulkarni, Umut Ozertem, and Rosie Jones. 2015. Characterizing and Predicting Voice Query Reformulation. In *CIKM*.

[12] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[13] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning Deep Structured Semantic Models for Web Search using Clickthrough Data. In *CIKM*.

[14] Jiepu Jiang, Wei Jeng, and Daqing He. 2013. How Do Users Respond to Voice Input Errors? Lexical and Phonetic Query Reformulation in Voice Search. In *SIGIR*.

[15] Thorsten Joachims. 2006. Training Linear SVMs in Linear Time. In *SIGKDD*.

[16] Rosie Jones and Kristina Lisa Klinkner. 2008. Beyond the Session Timeout: Automatic Hierarchical Segmentation of Search Topics in Query Logs. In *CIKM*.

[17] Evangelos Kanoulas, Ben Carterette, Mark Hall, Paul Clough, and Mark Sanderson. 2011. Overview of the TREC 2011 Session Track. In *TREC*.

[18] Jingjing Liu and Nicholas J. Belkin. 2010. Personalizing Information Retrieval for Multi-session Tasks: The Roles of Task Stage and Task Type. In *SIGIR*.

[19] Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. 2013. Linguistic Regularities in Continuous Space Word Representations. In *HLT/NAACL*.

[20] Bhaskar Mitra and Nick Craswell. 2017. Neural Models for Information Retrieval. *arXiv:1705.01509v1*.

[21] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to Match Using Local and Distributed Representations of Text for Web Search. In *WWW*.

[22] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *EMNLP*.

[23] Sundar Pichai. 2016. Google I/O Keynote.

[24] Jinfeng Rao, Hua He, and Jimmy Lin. 2016. Noise-Contrastive Estimation for Answer Selection with Deep Neural Networks. In *CIKM*.

[25] Johan Schalkwyk, Doug Beeferman, Françoise Beaufays, Bill Byrne, Ciprian Chelba, Mike Cohen, Maryam Kamvar, and Brian Strope. 2010. "Your Word is My Command": Google Search by Voice: A Case Study. In *Advances in Speech Recognition*.

[26] Jiulong Shan, Genqing Wu, Zhihong Hu, Xiliu Tang, Martin Jansche, and Pedro J. Moreno. 2010. Search by Voice in Mandarin Chinese. In *INTERSPEECH*.

[27] Milad Shokouhi, Rosie Jones, Umut Ozertem, Karthik Raghunathan, and Fernando Diaz. 2014. Mobile Query Reformulations. In *SIGIR*.

[28] Milad Shokouhi, Umut Ozertem, and Nick Craswell. 2016. Did You Say U2 or YouTube? Inferring Implicit Transcripts from Voice Search Logs. In *WWW*.

[29] Mark D. Smucker, James Allan, and Ben Carterette. 2007. A Comparison of Statistical Significance Tests for Information Retrieval Evaluation. In *CIKM*.

[30] Alessandro Sordoni, Yoshua Bengio, Hossein Vahabi, Christina Lioma, Jakob Grue Simonsen, and Jian-Yun Nie. 2015. A Hierarchical Recurrent Encoder-Decoder for Generative Context-Aware Query Suggestion. In *CIKM*.

[31] Greg Sterling. 2015. It's Official: Google Says More Searches Now On Mobile Than On Desktop. http://searchengineland.com/its-official-google-says-more-searches-now-on-mobile-than-on-desktop-220369. (2015). Accessed: 2017-08-16.

[32] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-RMSProp, Coursera: Neural Networks for Machine Learning. (2012).

[33] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. IRGAN: A Minimax Game for Unifying Generative and Discriminative Information Retrieval Models. In *SIGIR*.

[34] Ye-Yi Wang, Dong Yu, Yun-Cheng Ju, and Alex Acero. 2008. An Introduction to Voice Search. *IEEE Signal Processing Magazine* 25, 3 (2008), 29–38.

[35] Jeonghe Yi and Farzin Maghoul. 2011. Mobile Search Pattern Evolution: The Trend and the Impact of Voice Queries. In *WWW*.