Learning to Map Context-Dependent Sentences to Executable Formal Queries

Alane Suhr[†], **Srinivasan Iyer**[‡], **and Yoav Artzi**[†]

[†] Dept. of Computer Science and Cornell Tech, Cornell University, New York, NY
{suhr, yoav}@cs.cornell.edu
[‡] Paul G. Allen School of Computer Science & Engineering, Univ. of Washington, Seattle, WA
sviyer@cs.washington.edu

Abstract

We propose a context-dependent model to map utterances within an interaction to executable formal queries. To incorporate interaction history, the model maintains an interaction-level encoder that updates after each turn, and can copy sub-sequences of previously predicted queries during generation. Our approach combines implicit and explicit modeling of references between utterances. We evaluate our model on the ATIS flight planning interactions, and demonstrate the benefits of modeling context and explicit references.

1 Introduction

The meaning of conversational utterances depends strongly on the history of the interaction. Consider a user querying a flight database using natural language (Figure 1). Given a user utterance, the system must generate a query, execute it, and display results to the user, who then provides the next request. Key to correctly mapping utterances to executable queries is resolving references. For example, the second utterance implicitly depends on the first, and the reference *ones* in the third utterance explicitly refers to the response to the second utterance. Within an interactive system, this information needs to be composed with mentions of database entries (e.g., *Seattle*, *next Monday*) to generate a formal executable representation. In this paper, we propose encoder-decoder models that directly map user utterances to executable queries, while considering the history of the interaction, including both previous utterances and their generated queries.

Reasoning about how the meaning of an utterance depends on the history of the interaction is critical to correctly respond to user requests. As interactions progress, users may omit previously-mentioned constraints and entities, and an increasing portion of the utterance meaning must be derived from the interaction history. Figure 2 shows SQL queries for the utterances in Figure 1. As the interaction progresses, the majority of the generated query is derived from the interaction history (underlined), rather than from the current utterance. A key challenge is resolving what past information is incorporated and how. For example, in the figure, the second utterance depends on the set of flights defined by the first, while adding a new constraint. The third utterance further refines this set by adding a constraint to the constraints from both previous utterances. In contrast, the fourth utterance refers only to the first one, and skips the two utterances in between.[1] Correctly generating the fourth query requires understanding that the time constraint (*at 7pm*) can be ignored as it follows an airline constraint that has been replaced.

show me flights from seattle to boston next monday
[Table with 31 flights]

on american airlines
[Table with 5 flights]

which ones arrive at 7pm
[No flights returned]

show me delta flights
[Table with 5 flights]

. . .

Figure 1: An excerpt of an interaction from the ATIS flight planning system (Hemphill et al., 1990; Dahl et al., 1994). Each request is followed by a description of the system response.

We study complementary methods to enable this type of reasoning. The first set of methods implicitly reason about references by modifying the encoder-decoder architecture to encode information from previous utterances for generation decisions. We experiment with attending over previous utterances and using an interaction-level recurrent encoder. We also study explicitly maintaining a set of referents using segments from pre-

[1] An alternative explanation is that utterance four refers to utterance three, and deletes the time and airline constraints.

$\bar{x}_1$: *show me flights from seattle to boston next monday*

$\bar{y}_1$:
```
(SELECT DISTINCT flight.flight_id FROM flight WHERE (flight.from_airport IN (SELECT
airport_service.airport_code FROM airport_service WHERE airport_service.city_code IN (SELECT
city.city_code FROM city WHERE city.city_name = 'SEATTLE'))) AND (flight.to_airport IN (SELECT
airport_service.airport_code FROM airport_service WHERE airport_service.city_code IN (SELECT
city.city_code FROM city WHERE city.city_name = 'BOSTON'))) AND (flight.flight_days IN (SELECT
days.days_code FROM days WHERE days.day_name IN (SELECT date_day.day_name FROM date_day WHERE
date_day.year = 1993 AND date_day.month_number = 2 AND date_day.day_number = 8))));
```

$\bar{x}_2$: *on american airlines*

$\bar{y}_2$:
```
(SELECT DISTINCT flight.flight_id FROM flight WHERE (flight.airline_code = 'AA') AND (flight.from_airport
IN (SELECT airport_service.airport_code FROM airport_service WHERE airport_service.city_code IN (SELECT
city.city_code FROM city WHERE city.city_name = 'SEATTLE'))) AND (flight.to_airport IN (SELECT air
port_service.airport_code FROM airport_service WHERE airport_service.city_code IN (SELECT city.city_code
FROM city WHERE city.city_name = 'BOSTON'))) AND (flight.flight_days IN (SELECT days.days_code FROM
days WHERE days.day_name IN (SELECT date_day.day_name FROM date_day WHERE date_day.year = 1993 AND
date_day.month_number = 2 AND date_day.day_number = 8))));
```

$\bar{x}_3$: *which ones arrive at 7pm*

$\bar{y}_3$:
```
(SELECT DISTINCT flight.flight_id FROM flight WHERE (flight.airline_code = 'AA') AND (flight.from_airport
IN (SELECT airport_service.airport_code FROM airport_service WHERE airport_service.city_code IN (SELECT
city.city_code FROM city WHERE city.city_name = 'SEATTLE'))) AND (flight.to_airport IN (SELECT air
port_service.airport_code FROM airport_service WHERE airport_service.city_code IN (SELECT city.city_code
FROM city WHERE city.city_name = 'BOSTON'))) AND (flight.flight_days IN (SELECT days.days_code FROM
days WHERE days.day_name IN (SELECT date_day.day_name FROM date_day WHERE date_day.year = 1993 AND
date_day.month_number = 2 AND date_day.day_number = 8))) AND (flight.arrival_time = 1900));
```

$\bar{x}_4$: *show me delta flights*

$\bar{y}_4$:
```
(SELECT DISTINCT flight.flight_id FROM flight WHERE (flight.airline_code = 'DL') AND (flight.from_airport
IN (SELECT airport_service.airport_code FROM airport_service WHERE airport_service.city_code IN (SELECT
city.city_code FROM city WHERE city.city_name = 'SEATTLE'))) AND (flight.to_airport IN (SELECT air
port_service.airport_code FROM airport_service WHERE airport_service.city_code IN (SELECT city.city_code
FROM city WHERE city.city_name = 'BOSTON'))) AND (flight.flight_days IN (SELECT days.days_code FROM
days WHERE days.day_name IN (SELECT date_day.day_name FROM date_day WHERE date_day.year = 1993 AND
date_day.month_number = 2 AND date_day.day_number = 8))));
```

Figure 2: Annotated SQL queries ($\bar{y}_1$,...,$\bar{y}_4$) in the ATIS (Hemphill et al., 1990) domain for utterances ($\bar{x}_1$,...,$\bar{x}_4$) from Figure 1. Underlining (not part of the annotation) indicates segments originating from the interaction context.

vious queries. At each step, the decoder chooses whether to output a token or select a segment from the set, which is appended to the output in a single decoding step. In addition to enabling references to previously mentioned entities, sets, and constraints, this method also reduces the number of generation steps required, illustrated by the underlined segments in Figure 2. For example, the query $\bar{y}_2$ will require 17 steps instead of 94.

We evaluate our approach using the ATIS (Hemphill et al., 1990; Dahl et al., 1994) task, where a user interacts with a SQL flight database using natural language requests, and almost all queries require joins across multiple tables. In addition to reasoning about contextual phenomena, we design our system to effectively resolve database values, including resolution of time expressions (e.g., *next monday* in Figure 1) using an existing semantic parser. Our evaluation shows that reasoning about the history of the interaction is necessary, relatively increasing performance by 28.6% over a baseline with no access to this information, and that combining the implicit and explicit methods provides the best performance. Furthermore, our analysis shows that our full approach maintains its performance as interaction length increases, while the performance of systems without explicit modeling deteriorates. Our code is available at `https://github.com/clic-lab/atis`.

2 Technical Overview

Our goal is to map utterances in interactions to formal executable queries. We evaluate our approach with the ATIS corpus (Hemphill et al., 1990; Dahl et al., 1994), where users query a realistic flight planning system using natural language. The system responds by displaying tables and database entries. User utterances are mapped to SQL to query a complex database with 27 tables and 162K entries. 96.6% of the queries require joins of different tables. Section 7 describes ATIS.

Task Notation Let $\mathcal{I}$ be the set of all interactions, $\mathcal{X}$ the set of all utterances, and $\mathcal{Y}$ the set of all formal queries. A user utterance $\bar{x} \in \mathcal{X}$ of length $|\bar{x}|$ is a sequence $\langle x_1, \ldots, x_{|\bar{x}|}\rangle$, where each x_i is a natural language token. A formal query $\bar{y} \in \mathcal{Y}$ of length $|\bar{y}|$ is a sequence $\langle y_1, \ldots, y_{|\bar{y}|}\rangle$, where each y_i is a formal query token. An interaction $\bar{I} \in \mathcal{I}$ is a sequence of n utterance-query pairs $\langle(\bar{x}_1, \bar{y}_1), \ldots, (\bar{x}_n, \bar{y}_n)\rangle$ representing an interaction with n turns. To refer to indexed interactions and their content, we mark $\bar{I}^{(l)}$ as an interaction with index l, the i-th utterance and query in $\bar{I}^{(l)}$ as $\bar{x}_i^{(l)}$ and $\bar{y}_i^{(l)}$, and the j-th tokens in $\bar{x}_i^{(l)}$ and $\bar{y}_i^{(l)}$ as $x_{i,j}^{(l)}$ and $y_{i,j}^{(l)}$. At turn i, we denote the interaction history of length $i-1$ as $\bar{I}[:i-1] = \langle(\bar{x}_1, \bar{y}_1), \ldots, (\bar{x}_{i-1}, \bar{y}_{i-1})\rangle$. Given $\bar{I}[:i-1]$ and utterance $\bar{x}_i$ our goal is to generate $\bar{y}_i$, while considering both $\bar{x}_i$ and $\bar{I}[:i-1]$. Following the execution of $\bar{y}_i$, the interaction history at turn $i+1$ becomes $\bar{I}[:i] = \langle(\bar{x}_1, \bar{y}_1), \ldots, (\bar{x}_i, \bar{y}_i)\rangle$.

Model Our model is based on the recurrent neural network (RNN; Elman, 1990) encoder-decoder framework with attention (Cho et al., 2014; Sutskever et al., 2014; Bahdanau et al., 2015; Luong et al., 2015). We modify the model in three ways to reason about context from the interaction history by attending over previous utterances (Section 4.2), adding a turn-level recurrent encoder that updates after each turn (Section 4.3), and adding a mechanism to copy segments of queries from previous utterances (Section 4.4). We also design a scoring function to score values that are abstracted during pre-processing, including entities and times (Section 6). The full model selects between generating query tokens and copying complete segments from previous queries.

Learning We assume access to a training set that contains N interactions $\{\bar{I}^{(l)}\}_{l=1}^{N}$. We train using a token-level cross-entropy objective (Section 5). For models that use the turn-level encoder, we construct computational graphs for the entire interaction and back-propagate the loss for all queries together. Without the turn-level encoder, each utterance is processed separately.

Evaluation We evaluate using a test set $\{\bar{I}^{(l)}\}_{l=1}^{M}$ of M interactions. We measure the accuracy of each utterance for each test interaction against the annotated query and its execution result. For models that copy segments from previous queries, we evaluate using both predicted and gold previous queries.

3 Related Work

Mapping sentences to formal representations, commonly known as semantic parsing, has been studied extensively with linguistically-motivated compositional representations, including variable-free logic (e.g., Zelle and Mooney, 1996; Clarke et al., 2010), lambda calculus (e.g., Zettlemoyer and Collins, 2005; Artzi and Zettlemoyer, 2011; Kushman and Barzilay, 2013), and dependency-based compositional semantics (e.g., Liang et al., 2011; Berant et al., 2013). Recovering lambda-calculus representations was also studied with ATIS with focus on context-independent meaning using grammar-based approaches (Zettlemoyer and Collins, 2007; Kwiatkowski et al., 2011; Wang et al., 2014) and neural networks (Dong and Lapata, 2016; Jia and Liang, 2016).

Recovering context-independent executable representations has been receiving increasing attention. Mapping sentence in isolation to SQL queries has been studied with ATIS using statistical parsing (Popescu et al., 2004; Poon, 2013) and sequence-to-sequence models (Iyer et al., 2017). Generating executable programs was studied with other domains and formal languages (Giordani and Moschitti, 2012; Ling et al., 2016; Zhong et al., 2017; Xu et al., 2017). Recently, various approaches were proposed to use the formal language syntax to constrain the search space (Yin and Neubig, 2017; Rabinovich et al., 2017; Krishnamurthy et al., 2017; Cheng et al., 2017) making all outputs valid programs. These contributions are orthogonal to ours, and can be directly integrated into our decoder.

Generating context-dependent formal representations has received less attention. Miller et al. (1996) used ATIS and mapped utterances to semantic frames, which were then mapped to SQL queries. For learning, they required full supervision, including annotated parse trees and contextual dependencies.[2] Zettlemoyer and Collins (2009) addressed the problem with lambda calculus, using a semantic parser trained separately with context-independent data. In contrast, we generate executable formal queries and require only interaction query annotations for training.

Recovering context-dependent meaning was also studied with the SCONE (Long et al., 2016) and SequentialQA (Iyyer et al., 2017) corpora. We compare ATIS to these corpora in Section 7. Resolving explicit references, a part of our problem, has been studied as co-reference resolution (Ng, 2010). Context-dependent language understanding was also studied for dialogue systems, including with ATIS Tür et al. (2010). More recently, encoder-decoder methods were applied to dialogue systems (Peng et al., 2017; Li et al., 2017), including using hierarchical RNNs (Serban et al., 2016, 2017), an architecture related to our turn-level encoder. These approaches use slot-filling frames with limited expressivity, while we focus on the original representation of unconstrained SQL queries.

[2] Miller et al. (1996) provide limited details about their evaluation. Later work notes that they evaluate SQL query correctness (Zettlemoyer and Collins, 2009) with an accuracy of 78.4%, higher than our results. However, the lack of details (e.g., if the metric is strict or relaxed) makes comparison difficult. In addition, we use significantly less supervision, and re-split the data to avoid scenario bias (Section 7).

4 Context-dependent Model

We base our model on an encoder-decoder architecture with attention (Cho et al., 2014; Sutskever et al., 2014; Bahdanau et al., 2015; Luong et al., 2015). At each interaction turn i, given the current utterance $\bar{x}_i$ and the interaction history $\bar{I}[:i-1]$, the model generates the formal query $\bar{y}_i$. Figure 3 illustrates our architecture. We describe the base architecture, and gradually add components.

4.1 Base Encoder-Decoder Architecture

Our base architecture uses an encoder to process the user utterance $\bar{x}_i = \langle x_{i,1}, \ldots, x_{i,|\bar{x}_i|}\rangle$ and a decoder to generate the output query $\bar{y}_i$ token-by-token. This architecture does not observe the interaction history $\bar{I}[:i-1]$.

The encoder computes a hidden state $\mathbf{h}^E_j = [\mathbf{h}^{\overrightarrow{E}}_j; \mathbf{h}^{\overleftarrow{E}}_j]$ for each token $x_{i,j}$ using a bi-directional RNN. The forward RNN is defined by: [3]

$$\mathbf{h}^{\overrightarrow{E}}_j = \text{LSTM}^{\overrightarrow{E}}\left(\phi^x(x_{i,j}); \mathbf{h}^{\overrightarrow{E}}_{j-1}\right) \quad , \qquad (1)$$

where $\text{LSTM}^{\overrightarrow{E}}$ is a long short-term memory recurrence (LSTM; Hochreiter and Schmidhuber, 1997) and ϕ^x is a learned embedding function for input tokens. The backward RNN recurs in the opposite direction with separate parameters.

We generate the query with an RNN decoder. The decoder state at step k is:

$$\mathbf{h}^D_k = \text{LSTM}^D\left([\phi^y(y_{i,k-1}); \mathbf{c}_{k-1}]; \mathbf{h}^D_{k-1}\right) \quad ,$$

where LSTM^D is a two-layer LSTM recurrence, ϕ^y is a learned embedding function for query tokens, and $\mathbf{c}_k$ is an attention vector computed from the encoder states. $y_{i,0}$ is a special start token, and $\mathbf{c}_0$ is a zero-vector. The initial hidden state and cell memory of each layer are initialized as $\mathbf{h}^E_{|\bar{x}_i|}$ and $\mathbf{c}^E_{|\bar{x}_i|}$. The attention vector $\mathbf{c}_k$ is a weighted sum of the encoder hidden states:

$$\begin{aligned} s_k(j) &= \mathbf{h}^E_j \mathbf{W}^{\mathbf{A}} \mathbf{h}^D_k & (2) \\ \alpha_k &= \text{softmax}(\mathbf{s}_k) & (3) \\ \mathbf{c}_k &= \sum_{j=1}^{|\bar{x}_i|} \mathbf{h}^E_j \alpha_k(j) \quad , & (4) \end{aligned}$$

where $\mathbf{W}^{\mathbf{A}}$ is a learned matrix. The probabilities of output query tokens are computed as:

$$\begin{aligned} \mathbf{m}_k &= \tanh\left([\mathbf{h}^D_k; \mathbf{c}_k]\mathbf{W}^m\right) & (5) \\ P(y_{i,k} = w \mid \bar{x}_i, \bar{y}_{i,1:k-1}) &\propto \exp(\mathbf{m}_k \mathbf{W}^o_w + \mathbf{b}^o_w) & (6) \end{aligned}$$

where $\mathbf{W}^m$, $\mathbf{W}^o$, and $\mathbf{b}^o$ are learned.

4.2 Incorporating Recent History

We provide the model with the most recent interaction history by concatenating the previous h utterances $\langle \bar{x}_{i-h}, ..., \bar{x}_{i-1}\rangle$ with the current utterance in order, adding a special delimiter token between each utterance. The concatenated input provides the model access to previous utterances, but not to previously generated queries, or utterances that are more than h turns in the past. The architecture remains the same, except that the encoder and attention are computed over the concatenated sequence of tokens. The probability of an output query token is computed the same, but is now conditioned on the interaction history:

$$P(y_{i,k} = w \mid \bar{x}_i, \bar{y}_{i,1:k-1}, \bar{I}[:i-1]) \propto \exp(\mathbf{m}_k \mathbf{W}^o_w + \mathbf{b}^o_w) \quad . \qquad (7)$$

4.3 Turn-level Encoder

Concatenating recent utterances to provide access to recent history has computational drawbacks. The encoding of the utterance depends on its location in the concatenated string. This requires encoding all recent history for each new utterance, and does not allow re-use of computation between utterances during encoding. It also introduces a tradeoff between computation cost and expressivity: attending over the h previous utterances allows the decoder access to the information in these utterances when generating a query, but is computationally more expensive as h increases. We address this by encoding each utterance once. To account for the influence of the interaction history on utterance encoding, we maintain a discourse state encoding $\mathbf{h}^I_i$ computed with a turn-level recurrence, and use it during utterance encoding. The state is maintained and updated over the entire interaction. At turn i, this model has access to the complete prefix of the interaction $\bar{I}[:i-1]$ and the current request $\bar{x}_i$. In contrast, the concatenation-based encoder (Section 4.2) has access only to information from the previous h utterances. We also use positional encoding in the attention computation to account for the position of each utterance relative to the current utterance.

Formally, we modify Equation 1 to encode $\bar{x}_i$:

$$\mathbf{h}^{\overrightarrow{E}}_{i,j} = \text{LSTM}^{\overrightarrow{E}}\left(\left[\phi^x(x_{i,j}); \mathbf{h}^I_{i-1}\right]; \mathbf{h}^{\overrightarrow{E}}_{i,j-1}\right) \quad ,$$

where $\mathbf{h}^I_{i-1}$ is the discourse state following utterance $\bar{x}_{i-1}$. $\text{LSTM}^{\overleftarrow{E}}$ is modified analogously. In contrast to the concatenation-based model, the recurrence processes a single utterance. The dis-

[3] We omit the memory cell (often denoted as $\mathbf{c}_j$) from all LSTM descriptions. We use only the LSTM hidden state $\mathbf{h}_j$ in other parts of the architecture unless explicitly noted.

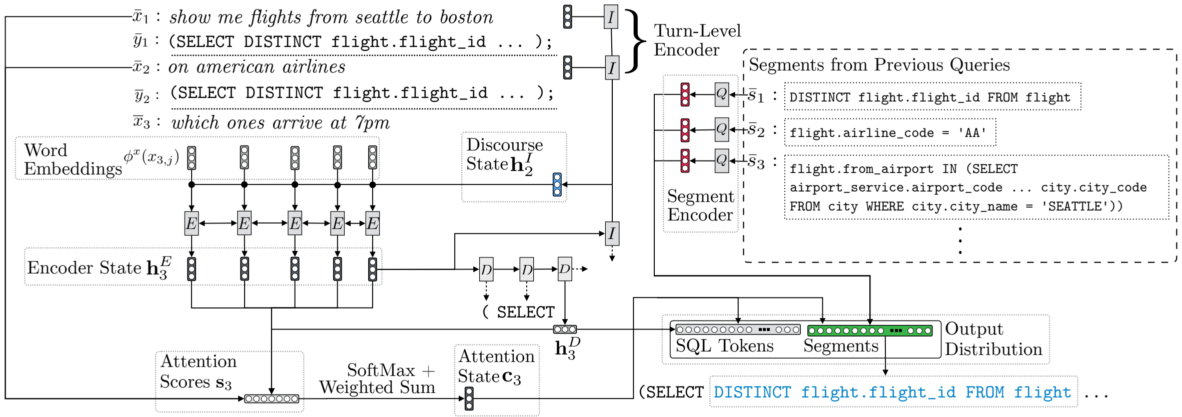

$\bar{x}_1$: show me flights from seattle to boston
$\bar{y}_1$: (SELECT DISTINCT flight.flight_id ...);
$\bar{x}_2$: on american airlines
$\bar{y}_2$: (SELECT DISTINCT flight.flight_id ...);
$\bar{x}_3$: which ones arrive at 7pm
Word Embeddings
$\phi^x(x_{3,j})$
Discourse State $\mathbf{h}_2^I$
Encoder State $\mathbf{h}_3^E$
Turn-Level Encoder
Segments from Previous Queries
$\bar{s}_1$: DISTINCT flight.flight_id FROM flight
$\bar{s}_2$: flight.airline_code = 'AA'
$\bar{s}_3$: flight.from_airport IN (SELECT airport_service.airport_code ... city.city_code FROM city WHERE city.city_name = 'SEATTLE'))
Segment Encoder
I
E
D
Q
(SELECT
$\mathbf{h}_3^D$
SQL Tokens
Segments
Output Distribution
Attention Scores $\mathbf{s}_3$
SoftMax + Weighted Sum
Attention State $\mathbf{c}_3$
(SELECT DISTINCT flight.flight_id FROM flight ...

Figure 3: Illustration of the model architecture during the third decoding step while processing the instruction *which ones arrive at 7pm* from the interaction in Figure 2. The current discourse state $\mathbf{h}_2^I$ is used to encode the current utterance $\bar{x}_3$ (Section 4.3). Query segments from previous queries are encoded into vector representations (Section 4.4). In each generation step, the decoder attends over the previous and current utterances, and a probability distribution is computed over SQL tokens and query segments. Here, segment $\bar{s}_1$ is selected.

course state $\mathbf{h}^I_i$ is computed as

$$\mathbf{h}^I_i = \text{LSTM}^I\left(\mathbf{h}^E_{i,|\bar{x}_i|}; \mathbf{h}^I_{i-1}\right) .$$

Similar to the concatenation-based model, we attend over the current utterance and the h previous utterances. We add relative position embeddings ϕ^I to each hidden state. These embeddings are learned for each possible distance $0, \dots, h-1$ from the current utterance. We modify Equation 2 to index over both utterances and tokens:

$$s_k(t,j) = \left[\mathbf{h}^E_{t,j}; \phi^I(i-t)\right] \mathbf{W}^{\mathbf{A}}\mathbf{h}^D_k . \tag{8}$$

In contrast to the concatenation model, without position embeddings, the attention computation has no indication of the utterance position, as our ablation shows in Section 8. The attention distribution is computed as in Equation 3, and normalized across all utterances. The position embedding is also used to compute the context vector $\mathbf{c}_k$:

$$\mathbf{c}_k = \sum_{t=i-h}^{i} \sum_{j=1}^{|\bar{x}_t|} \left[\mathbf{h}^E_{t,j}; \phi^I(i-t)\right] \alpha_k(t,j) .$$

4.4 Copying Query Segments

The discourse state and attention over previous utterances allow the model to consider the interaction history when generating queries. However, we observe that context-dependent reasoning often requires generating sequences that were generated in previous turns. Figure 2 shows how segments (underlined) extracted from previous utterances are predominant in later queries. To take advantage of what was previously generated, we add copying of complete segments from previous queries by expanding the set of outputs at each generation step. This mechanism explicitly models references, reduces the number of steps required to generate queries, and provides an interpretable view of what parts of a query originate in context. Figure 3 illustrates this architecture.

Extracting Segments Given the interaction history $\bar{I}[: i-1]$, we construct the set of segments S_{i-1} by deterministically extracting subtrees from previously generated queries.[4] In our data, we extract 13 ± 5.9 ($\mu \pm \sigma$) segments for each annotated query. Each segment $\bar{s} \in S_{i-1}$ is a tuple $\langle a, b, l, r\rangle$, where a and b are the indices of the first and most recent queries, $\bar{y}_a$ and $\bar{y}_b$, in the interaction that contain the segment. l and r are the start and end indices of the segment in $\bar{y}_b$.

Encoding Segments We represent a segment $\bar{s} = \langle a, b, l, r\rangle$ using the hidden states of an RNN encoding of the query $\bar{y}_b$. The hidden states $\langle \mathbf{h}^Q{}_1, ..., \mathbf{h}^Q{}_{|\bar{y}_b|}\rangle$ are computed using a bi-directional LSTM RNN similar to the utterance encoder (Equation 1), except using separate LSTM parameters and ϕ^y to embed the query tokens. The embedded representation of a segment is a concatenation of the hidden states at the segment endpoints and an embedding of the relative position of the utterance where it appears first:

$$\mathbf{h}^S = \left[\mathbf{h}^Q_l; \mathbf{h}^Q_r; \phi^g(\min(g, i-a))\right] ,$$

where ϕ^g is a learned embedding function of the position of the initial query $\bar{y}_a$ relative to the current turn index i. We learn an embedding for each relative position that is smaller than g, and use the same embedding for all other positions.

Generation with Segments At each generation step, the decoder selects between a single query token or a segment. When a segment is selected, it

[4] The extraction process is detailed in Appendix B.

is appended to the generated query, an embedded segment representation for the next step is computed, and generation continues. The probability of a segment $\bar{s} = \langle a, b, l, r \rangle$ at decoding step k is:

$$P(y_{i,k} = \bar{s} \mid \bar{x}_i, \bar{y}_{i,1:k-1}, \bar{I}[: i-1]) \propto \exp\left(\mathbf{m}_k \mathbf{W}^{S} \mathbf{h}^{S}\right) \quad , \tag{9}$$

where $\mathbf{m}_k$ is computed in Equation 5 and $\mathbf{W}^{S}$ is a learned matrix. To simplify the notation, we assign the segment to a single output token. The output probabilities (Equations 7 and 9) are normalized together to a single probability distribution. When a segment is selected, the embedding used as input for the next generation step is a bag-of-words encoding of the segment. We extend the output token function ϕ^y to take segments:

$$\phi^y(\bar{s} = \langle a, b, l, r \rangle) = \frac{1}{r-l} \sum_{k=l}^{r} \phi^y\left(y_{b,k}\right) \quad .$$

The recursion in ϕ^y is limited to depth one because segments do not contain other segments.

4.5 Inference with Full Model

Given an utterance $\bar{x}_i$ and the history of interaction $\bar{I}[: i-1]$, we generate the query $\bar{y}_i$. An interaction starts with the user providing the first utterance $\bar{x}_1$. The utterance is encoded using the initial discourse state $\mathbf{h}^I_0$, the discourse state $\mathbf{h}^I_1$ is computed, the query $\bar{y}_1$ is generated, and the set of segments S_1 is created. The initial discourse state $\mathbf{h}^I_0$ is learned, and the set of segments S_0 used when generating $\bar{y}_1$ is the empty set. The attention is computed only over the first utterance because no previous utterances exist. The user then provides the next utterance or concludes the interaction. At turn i, the utterance $\bar{x}_i$ is encoded using the discourse state $\mathbf{h}^I_{i-1}$, the discourse state $\mathbf{h}^I_i$ is computed, and the query $\bar{y}_i$ is generated using the set of segments S_{i-1}. The model has no access to future utterances. We use greedy inference for generation. Figure 3 illustrates a single decoding step.

5 Learning

We assume access to a training set of N interactions $\{\bar{I}^{(l)}\}_{l=1}^{N}$. Given an interaction $\bar{I}^{(l)}$, each utterance $\bar{x}_i^{(l)}$ where $1 \leq i \leq |\bar{I}^{(l)}|$, is paired with an annotated query $\bar{y}_i^{(l)}$. The set of segments from previous utterances is deterministically extracted from the annotated queries during learning. However, the data does not indicate what parts of each query originate in segments copied from previous utterances. We adopt a simple approach and heuristically identify context-dependent segments based on entities that appear in the utterance and the query.[5] Once we identify a segment in the annotated query, we replace it with a unique placeholder token, and it appears to the learning algorithm as a single generation decision. Treating this decision as latent is an important direction for future work. Given the segment copy decisions, we minimize the token cross-entropy loss:

$$\mathcal{L}(y_{i,k}^{(l)}) = -\log P\left(y_{i,k}^{(l)} \mid \bar{x}_i^{(l)}, \bar{y}_{i,1:k-1}^{(l)}, \bar{I}^{(l)}[: i-1]\right) \quad ,$$

where k is the index of the output token. The base and recent-history encoders (Sections 4.1 and 4.2) can be trained by processing each utterance separately. For these models, given a mini-batch $\mathcal{B}$ of utterances, each identified by an interaction-utterance index pair, the loss is the mean token loss

$$\mathcal{L} = \frac{1}{\sum_{(i,j)\in\mathcal{B}} |\bar{y}_i^{(j)}|} \sum_{(i,j)\in\mathcal{B}} \sum_{k=1}^{|\bar{y}_i^{(j)}|} \mathcal{L}(y_{i,k}^{(j)}) \quad .$$

The turn-level encoder (Section 4.3) requires building a computation graph for the entire interaction. We update the model parameters for each interaction. The interaction loss is

$$\mathcal{L} = \frac{n}{B} \frac{1}{\sum_{i=1}^{n} |\bar{y}_i^{(j)}|} \sum_{i=1}^{n} \sum_{k=1}^{|\bar{y}_i^{(j)}|} \mathcal{L}(y_{i,k}^{(j)}) \quad ,$$

where B is the batch size, and $\frac{n}{B}$ re-normalizes the loss so the gradient magnitude is not dependent on the number of utterances in the interaction. Our ablations ($-$batch re-weight in Table 2) shows the importance of this term. For both cases, we use teacher forcing (Williams and Zipser, 1989).

6 Reasoning with Anonymized Tokens

An important practical consideration for generation in ATIS and other database domains is reasoning about database values, such as entities, times, and dates. For example, the first utterance in Figure 2 includes two entities and a date reference. With limited data, learning to both reason about a large number of entities and to resolve dates are challenging for neural network models. Following previous work (Dong and Lapata, 2016; Iyer et al., 2017), we address this with anonymization, where the data is pre- and post-processed to abstract over tokens that can be heuristically resolved to tokens in the query language. In contrast to previous work, we design a special scoring function to anonymized tokens to reflect how they are used in the input utterances. Figure 4 illustrates preprocessing in ATIS. For example, we use a temporal semantic parser to resolve dates (e.g., *next*

[5] Appendix B describes the alignment process.

Original utterance and query:

$\bar{x}_1$: *show me flights from seattle to boston next monday*

```
ȳ1: ( SELECT DISTINCT flight.flight_id ...
   city.city_name = 'SEATTLE' ...  city.city_name
   = 'BOSTON' ...  date_day.year = 1993 AND
   date_day.month_number = 2 AND date_day.day_number
   = 8 ...
```

Anonymized utterance and query:

$\bar{x}'_1$: *show me flights from CITY#1 to CITY#2 DAY#1 MONTH#1 YEAR#1*

```
ȳ'1: ( SELECT DISTINCT flight.flight_id ...
   city.city_name = CITY#1 ...  city.city_name
   = CITY#2 ...  date_day.year = YEAR#1
   AND date_day.month_number = MONTH#1 AND
   date_day.day_number = DAY#1 ...
```

Anonymization mapping:

CITY#1	'SEATTLE'	MONTH#1	2
CITY#2	'BOSTON'	YEAR#1	1993
DAY#1	8		

Figure 4: An example of date and entity anonymization pre-processing for $\bar{x}_1$ and $\bar{y}_1$ in Figure 2.

Monday) and replace them with day, month, and year placeholders. To anonymize database entries, we use a dictionary compiled from the database (e.g., to map *Seattle* to SEATTLE). The full details of the anonymization procedure are provided in the supplementary material. Following pre-processing, the model reasons about encoding and generation of anonymized tokens (e.g., CITY#1) in addition to regular output tokens and query segments from the interaction history. Anonymized tokens are typed (e.g., CITY), map to a token in the query language (e.g., 'BOSTON'), and appear both in input utterances and generated queries.

We modify our encoder and decoder embedding functions (ϕ^x and ϕ^y) to map anonymized tokens to the embeddings of their types (e.g., CITY). The type embeddings in ϕ^x and ϕ^y are separate. Using the types only, while ignoring the indices, avoids learning biases that arise from the arbitrary ordering of the tokens in the training data. However, it does not allow distinguishing between entries with the same type for generation decisions; for example, the common case where multiple cities are mentioned in an interaction. We address this by scoring anonymized token based on the magnitude of attention assigned to them at generation step k. The attention magnitude is computed from the encoder hidden states. This computation considers both the decoder state and the location of the anonymized tokens in the input utterances to account for how they are used in the interaction. The probability of an anonymized token w at generation step k is

$$P(y_{i,k} = w \mid \bar{x}_i, \bar{y}_{i,1:k-1}, \bar{I}[:i-1]) \propto \sum_{t=i-h}^{i} \sum_{j=1}^{|\bar{x}_t|} \left(\exp\left(s_k\left(t,j\right)\right)\right)$$

where $s_k\left(t,j\right)$ is the attention score computed in Equation 8. This probability is normalized together with the probabilities in Equations 7 and 9 to form the complete output probability.

Mean/max utterances per interaction	7.0 / 64
Mean/max tokens per utterance	10.2 / 47
Mean/max token per SQL query	102.9 / 1286
Input vocabulary size	1582
Output vocabulary size	982

Table 1: ATIS data statistics.

7 Experimental Setup

Hyperparameters, architecture details, and other experimental choices are detailed in Appendix C.

Data We use ATIS (Hemphill et al., 1990; Dahl et al., 1994) to evaluate our approach. The data was originally collected using wizard-of-oz experiments, and annotated with SQL queries. Each interaction was based on a scenario given to a user. We observed that the original data split shares scenarios between the train, development, and test splits. This introduces biases, where travel patterns that appeared during training repeat in testing. For example, a model trained on the original data split often correctly resolves the exact referenced by *on Saturday* with no pre-processing or access to the document date. We evaluate this overfitting empirically in Appendix D.1. We re-split the data to avoid this bias. We evenly distribute scenarios across splits so that each split contains both scenarios with many and few representative interactions. The new split follows the original split sizes with 1148/380/130 train/dev/test interactions. Table 1 shows data statistics. The system uses a SQL database of 27 tables and 162K entries. 96.6% of the queries require at least one join, and 93% at least two joins. The most related work on ATIS to ours is Miller et al. (1996), which we discuss in Section 3.

The most related corpora to ATIS are SCONE (Long et al., 2016) and SequentialQA (Iyyer et al., 2017). SCONE (Long et al., 2016) contains micro-domains consisting of stack- or list-like elements. The formal representation is linguistically-motivated and the majority of queries include a single binary predicate. All interactions include five turns. SequentialQA (Iyyer et al., 2017) contains sequences of questions on a single Wikipedia table. Interactions are on average 2.9 turns long, and were created by re-phrasing a question from a context-independent corpus (Pasupat and Liang, 2015). In contrast, ATIS uses a significantly larger database, requires generating complex queries with multiple joins, includes longer interactions, and was collected through interaction with users. Appendix E.1 contains detailed analysis of the contextual phenomena observed in ATIS.

Pre-processing We pre-process the data to identify and anonymize entities (e.g., cities), numbers, times, and dates. We use string matching heuristics to identify entities and numbers, and identify and resolve times and dates using UWTime (Lee et al., 2014). When resolving dates we use the original interaction date as the document time. Appendix A details this process.

Metrics We evaluate using query accuracy, strict denotation accuracy, and relaxed denotation accuracy. Query accuracy is the percentage of predicted queries that match the reference query. Strict denotation accuracy is the percentage of predicted queries that execute to exactly the same table as the reference query. In contrast to strict, relaxed gives credit to a prediction query that fails to execute if the reference table is empty. In cases when the utterance is ambiguous and there are multiple gold queries, we consider the query or table correct if they match any of the gold labels.

Systems We evaluate four systems: (a) SEQ2SEQ-0: the baseline encoder-decoder model (Section 4.1); (b) SEQ2SEQ-H: encoder-decoder with attention on current and previous utterances (Section 4.2); (c) S2S+ANON: encoder-decoder with attention on previous utterances and anonymization scoring (Section 6); and (d) FULL: the complete approach including segment copying (Section 4.4). For FULL, we evaluate with predicted and gold (FULL-GOLD) previous queries, and without attention on previous utterances (FULL-0). All models except SEQ2SEQ-0 and FULL-0 use $h = 3$ previous utterances. We limit segment copying to segments that appear in the most recent query only.[6] Unless specifically ablated, all experiments use pre-processing.

8 Results

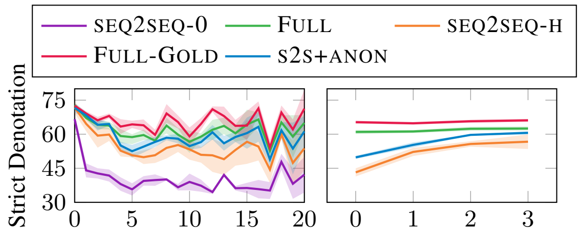
Table 2 shows development and test results. We run each experiment five times and report mean and standard deviation. The main metric we focus on is strict denotation accuracy. The relatively low performance of SEQ2SEQ-0 demonstrates the need for context in this task. Attending on recent history significantly increases performance. Both SEQ2SEQ models score anonymized tokens as regular vocabulary tokens. Adding anonymized token scoring further increases performance (S2S+ANON). FULL-0 and FULL add segment copying and the turn-level encoder. The relatively high performance of FULL-0 shows that substituting segment copying with attention maintains and even improves the system effectiveness. However, the best performance is provided with FULL, which combines both. This shows the benefit of redundancy in accessing contextual information. Unlike the other systems, both FULL and FULL-0 suffer from cascading errors due to selecting query segments from previously incorrect predictions. The higher FULL-GOLD performance illustrates the influence of error propagation. While part of this error can be mitigated by having both attention and segment copying, this behavior is unlikely to be learned from supervised learning, where errors are never observed.

Model	**Query**	**Denotation**	
		Relaxed	**Strict**
Development Results			
SEQ2SEQ-0	28.7±1.7	48.8±1.4	43.2±1.8
SEQ2SEQ-H	35.1±2.2	59.4±2.4	56.7±3.2
S2S+ANON	37.6±0.7	61.6±0.7	60.6±0.7
FULL-0	36.3±0.5	61.5±0.8	61.0±0.9
FULL	37.5±0.9	63.0±0.7	**62.5**±0.9
– turn-level enc.	37.4±1.5	62.1±2.5	61.4±2.7
– batch re-weight	36.4±0.6	61.8±0.3	61.5±0.4
– input pos. embs.	33.3±0.2	57.9±0.8	57.4±0.8
– query segments	36.0±0.9	59.5±1.3	58.3±1.4
– anon. scoring	35.7±0.5	60.8±1.1	60.0±1.0
– pre-processing	26.4±6.1	53.3±8.6	53.0±8.5
FULL-GOLD	42.1±0.8	66.6±0.7	66.1±0.7
Test Results			
SEQ2SEQ-0	35.7±1.5	56.4±1.1	53.8±1.0
SEQ2SEQ-H	42.2±2.0	66.6±3.2	65.8±3.4
S2S+ANON	44.0±1.2	69.3±1.0	68.6±1.1
FULL-0	43.1±1.3	67.8±1.6	67.2±1.6
FULL	43.6±1.0	69.3±0.8	**69.2**±0.8
FULL-GOLD	47.4±1.3	72.3±0.5	72.0±0.5

Table 2: Mean and standard deviation development and test results, including ablations on the FULL model.

Ablations show that all components contribute to the system performance. Performance drops when using a concatenation-based encoder instead of the turn-level encoder (−turn-level enc.; Section 4.3). Using batch-reweighting (−batch-reweight; Section 5) and input position embeddings (−input pos. embs.; Section 4.3) are critical to the performance of the turn-level encoder. Removing copying of query segments from the interaction history lowers performance

[6] While we only use segments from the most recent query, they often appear for the first time much earlier in the interaction, which influences their absolute position value a.



(a) Interaction turns (b) History length h

Figure 5: Mean development strict denotation accuracy as function of turns and h.

Model	Query	Denotation	
		Relaxed	Strict
Development Results			
S2S+ANON	44.4±1.2	69.9±0.3	68.9±0.3
FULL	42.8±0.2	68.8±0.2	68.4±0.2
FULL-GOLD	47.5±0.2	71.5±0.4	70.7±0.6
Test Results			
S2S+ANON	43.9±0.3	67.4±1.0	67.2±1.0
FULL	44.3±0.2	66.3±0.3	66.3±0.4
FULL-GOLD	47.2±0.3	68.2±0.5	67.9±0.4

Table 3: Results on the original split of the data.

(−query segments; Section 4.4). Treating indexed anonymized tokens as regular tokens, rather than using attention-based scoring and type embeddings, lowers performance (−anon. scoring; Section 6). Finally, pre-processing, which includes anonymization, is critical (−pre-processing).

Figure 5(a) shows the performance as interactions progress. All systems show a drop in performance after the first utterance, which is always context-independent. As expected, SEQ2SEQ-0 shows the biggest drop. The FULL approach is the most stable as the interaction progresses.

Figure 5(b) shows the performance as we decrease the number of previous utterances used for attention h. Without the turn-level encoder and segment copying (SEQ2SEQ-H and S2S+ANON), performance decreases significantly as h decreases. In contrast, the FULL model shows a smaller decrease (1.5%). Appendix E.2 includes attention analysis demonstrating the importance of previous-utterance attention. However, attending on fewer utterances improves inference speed: FULL-0 is 30% faster than FULL.

Finally, while we re-split the data due to scenario sharing between train and test early in development and used this split only for development, we also evaluate on the original split (Table 3). We report mean and standard deviation over three trials. The high performance of S2S+ANON potentially indicates it benefits more from the differences between the splitting procedures.

9 Analysis

We analyze errors made by the full model on thirty development interactions. When analyzing the output of FULL, we focus on error propagation and analyze predictions that resulted in an incorrect table when using FULL, but a correct table when using FULL-GOLD. 56.7% are due to selection of a segment that contained an incorrect constraint. 43.4% of the errors are caused by a necessary segment missing during generation. 93.0% of all predictions are valid SQL and follow the database schema. We also analyze the errors of FULL-GOLD. We observe that 30.0% of errors are due to generating constraints that were not mentioned by the user. Other common errors include generating relevant constraints with incorrect values (23.3%) and missing constraints (23.3%).

We also evaluate our model's ability to recover long-distance references while constraints are added, changed, or removed, and when target attributes change. Appendix E.3 includes the analysis details. In general, the model resolves references well. However, it fails to recover constraints mentioned in the past following a user focus state change (Grosz and Sidner, 1986).

10 Discussion

We study models that recover context-dependent executable representations from user utterances by reasoning about interaction history. We observe that our segment-copying models suffer from error propagation when extracting segments from previously-generated queries. This could be mitigated by training a model to ignore erroneous segments, and recover by relying on attention for generation. However, because supervised learning does not expose the model to erroneous states, a different learning approach is required. Our analysis demonstrates that our model is relatively insensitive to interaction length, and is able to recover both explicit and implicit references to previously-mentioned entities and constraints. Further study of user focus change is required, an important phenomenon that is relatively rare in ATIS.

Acknowledgements

This research was supported by the NSF (CRII-1656998), Schmidt Sciences, a gift from Google, and cloud computing credits from Amazon. We thank Valts Blukis, Luke Zettlemoyer, and the anonymous reviewers for their helpful comments.

References

Yoav Artzi and Luke Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. http://www.aclweb.org/anthology/D11-1039.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. http://www.aclweb.org/anthology/D13-1160.

Jianpeng Cheng, Siva Reddy, Vijay Saraswat, and Mirella Lapata. 2017. Learning structured natural language representations for semantic parsing. In *Proceedings Association for Computational Linguistics*. https://doi.org/10.18653/v1/P17-1005.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. https://doi.org/10.3115/v1/D14-1179.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world's response. In *Proceedings of the Conference on Computational Natural Language Learning*. http://www.aclweb.org/anthology/W10-2903.

Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the ATIS task: The ATIS-3 corpus. In *Proceedings of the Workshop on Human Language Technology*. http://www.aclweb.org/anthology/H94-1010.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the Association for Computational Linguistics*. https://doi.org/10.18653/v1/P16-1004.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science* 14:179–211.

Alessandra Giordani and Alessandro Moschitti. 2012. Translating questions to SQL queries with generative parsers discriminatively reranked. In *Conference on Computational Linguistics*. http://www.aclweb.org/anthology/C12-2040.

Barbara J. Grosz and Candace L. Sidner. 1986. Attention, intentions, and the structure of discourse. *Computational Linguistics* 12(3). http://www.aclweb.org/anthology/J86-3001.

Charles T. Hemphill, John J. Godfrey, and George R. Doddington. 1990. The ATIS spoken language systems pilot corpus. In *Proceedings of the DARPA Speech and Natural Language Workshop*. http://www.aclweb.org/anthology/H90-1021.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the the Association for Computational Linguistics*. https://doi.org/10.18653/v1/P17-1089.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based neural structured learning for sequential question answering. In *Proceedings of the Association for Computational Linguistics*. https://doi.org/10.18653/v1/P17-1167.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the Association for Computational Linguistics*. https://doi.org/10.18653/v1/P16-1002.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of theConference on Empirical Methods in Natural Language Processing*. https://www.aclweb.org/anthology/D17-1160.

Nate Kushman and Regina Barzilay. 2013. Using semantic unification to generate regular expressions from natural language. In *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. http://www.aclweb.org/anthology/N13-1103.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*. http://www.aclweb.org/anthology/D11-1140.

Kenton Lee, Yoav Artzi, Jesse Dodge, and Luke Zettlemoyer. 2014. Context-dependent semantic parsing for time expressions. In *Proceedings of the Association for Computational Linguistics*. https://doi.org/10.3115/v1/P14-1135.

Xiujun Li, Yun-Nung Chen, Lihong Li, and Jianfeng Gao. 2017. End-to-end task-completion neural dialogue systems. *CoRR* abs/1703.01008.

Percy Liang, Michael Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of the Association for Computational Linguistics: Human Language Technologies*. http://www.aclweb.org/anthology/P11-1060.

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomás Kociský, Fumin Wang, and Andrew Senior. 2016. Latent predictor networks for code generation. *CoRR* abs/1603.06744.

Reginald Long, Panupong Pasupat, and Percy Liang. 2016. Simpler context-dependent logical forms via model projections. In *Proceedings of the Association for Computational Linguistics*. https://doi.org/10.18653/v1/P16-1138.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. https://doi.org/10.18653/v1/D15-1166.

Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. 1996. A fully statistical approach to natural language interfaces. In *Proceedings of the Association for Computational Linguistics*. http://www.aclweb.org/anthology/P96-1008.

Vincent Ng. 2010. Supervised noun phrase coreference research: The first fifteen years. In *Proceedings of the Association for Computational Linguistics*. http://www.aclweb.org/anthology/P10-1142.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the Association for Computational Linguistics*. https://doi.org/10.3115/v1/P15-1142.

Baolin Peng, Xiujun Li, Lihong Li, Jianfeng Gao, Asli Celikyilmaz, Sungjin Lee, and Kam-Fai Wong. 2017. Composite task-completion dialogue policy learning via hierarchical deep reinforcement learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. https://aclanthology.info/pdf/D/D17/D17-1236.pdf.

Hoifung Poon. 2013. Grounded unsupervised semantic parsing. In *Proceedings of the Association for Computational Linguistics*. http://www.aclweb.org/anthology/P13-1092.

Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *International Conference on Computational Linguistics*. http://www.aclweb.org/anthology/C04-1021.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the Association for Computational Linguistics*. https://doi.org/10.18653/v1/P17-1105.

Iulian Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Association for the Advancement of Artificial Intelligence*.

Iulian Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio. 2017. A hierarchical latent variable encoder-decoder model for generating dialogues. In *Association for the Advancement of Artificial Intelligence*.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Neural Information Processing Systems*.

Gökhan Tür, Dilek Z. Hakkani-Tür, and Larry P. Heck. 2010. What is left to be understood in ATIS? In *Proceedings of the Spoken Language Technology Workshop*.

Adrienne Wang, Tom Kwiatkowski, and Luke Zettlemoyer. 2014. Morpho-syntactic lexical generalization for CCG semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. https://doi.org/10.3115/v1/D14-1135.

Ronald J. Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation* 1:270–280.

Xiaojun Xu, Chang Liu, and Dawn Xiaodong Song. 2017. SQLNet: Generating structured queries from natural language without reinforcement learning. *CoRR* abs/1711.04436.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the Association for Computational Linguistics*. https://doi.org/10.18653/v1/P17-1041.

John M. Zelle and Raymond J. Mooney. 1996. Comparative results on using inductive logic programming for corpus-based parser construction. In *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Luke S. Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. http://www.aclweb.org/anthology/D07-1071.

Luke S. Zettlemoyer and Michael Collins. 2009. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Joint Conference of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing of the AFNLP*. http://www.aclweb.org/anthology/P09-1110.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *CoRR* abs/1709.00103.

A Data Processing

Date and Number Resolution We replace instances of spelled-out multi-digit numbers in the original data (e.g., *flight two three five*) with a numerical representation (e.g., *flight 235*). We resolve expressions containing references to dates using UWTime (Lee et al., 2014). Interactions in ATIS are annotated with the date they took place, which we use as document time. We use the *newswire* annotator in UWTime to annotate each utterance in ATIS with date tags, and add one week to any predicted dates that occur before the document date. This heuristic follows the assumption that users always ask for information in the future. UWTime is able to predict dates contained in the gold queries in the training data with approximately 70% accuracy. Without using interaction dates and resolving date expressions, the model would not be able to generate correct dates, unless through overfitting to the training set. Previous work on ATIS addresses the problem of resolving relative date expressions by modifying the output queries. For example, Zettlemoyer and Collins (2009) add logical constants such as *tomorrow* to the lambda-calculus lexicon. To the best of our knowledge, no previous context-dependent work on ATIS uses interaction dates to recover the referents of date expressions.

Entity Anonymization We replace known entities in user utterances and SQL queries with anonymized tokens. Using the database, we generate a set of known entities and their natural language and SQL forms, for example the set of city names from the `city` table. When anonymizing an example, we first identify entities that occur in the input sequence, and replace each with a unique anonymized token, using the same token for entities that occur multiple times. We also identify numerical constants, which include flight numbers and times, as entities in the input sequence. This process gives us a set of entities in the input sequence and their corresponding anonymized tokens. During training, we replace any tokens in the gold SQL query that are in this set with the appropriate anonymized token. Entity anonymization is separate of entity scoring, described in Section 6, which computes scores for generating anonymized tokens independently of generating raw SQL tokens. When entity scoring is ablated in our experiments, entity anonymization is still applied as a pre-processing technique, but anonymized tokens with indices are used as regular members of the input and output vocabularies.

Post-processing After generating a query but before evaluating it against the SQL database or comparing with the gold labels, we post-process it. We de-anonymize all generated anonymized tokens using the anonymization set extracted from the input sequences, and correct mismatched parentheses by adding closing parenthesis at the end of the query.

B Copying Segments

Segment Extraction from Previous Queries To construct $S(\bar{y})$, we deterministically extract subtrees of the SQL parse tree from $\bar{y}$, as well as `SELECT` statements containing special modifiers like `MIN`. We use the `sqlparse` package to construct a tree, and recursively traverse its subtrees. We consider all subtrees of a `WHERE` clause. We separate children of conjunctions into distinct subtrees. When evaluating with predicted queries, we extract segments from the most recent prediction that has correct syntax and follows the database structure.

Alignment of Segments with Gold Queries During training, we align the set of extracted segments $S(\bar{y})$ with the gold query to construct a new query that contains references to extracted segments. We first extract known entities, e.g. city names, from the current utterance $\bar{x}_i$, using the entity set described above. We greedily substitute the longest extracted segments in $S(\bar{y})$ first. If segment $\bar{s}$ is a subsequence of the gold query $\bar{y}_i$, we replace that subsequence with a reference to it. We do not replace the subsequence if it contains one of the entities extracted from the input sequence; entities mentioned in the current utterance should be explicitly generated in the current prediction, as these are most likely not references to previous constraints.

C Implementation Details

Learning We use the ADAM optimizer (Kingma and Ba, 2014) with an initial global learning rate of 0.001. The batch size $B = 16$. We use patience as a stopping mechanism, with an initial patience of 10 epochs. We compute loss and token-level accuracy on a held-out validation set after each epoch. This set includes 5% of the training data, and when using our split of the dataset, does not contain scenarios that are present in the remaining 95% of the data. We use the same validation set across all of our experiments. After an epoch, if token-level loss on the validation set increases since the previous epoch, the global learning rate is multiplied by 0.8. When token-level accuracy on the validation set increases to a maximum value during training, we multiply the current patience by 1.01. During training, we apply dropout with probability 0.5 after the first decoder LSTM layer, and after computing $\mathbf{m}_k$ at each decoding step. The model parameters used to evaluate on the development and test sets are those that yielded the highest string-level validation accuracy.

During training, when multiple gold labels are present, we use the shortest. Loss is not computed for utterance-query pairs if, after pre-processing and query segment alignment, the gold label contains more than 200 tokens. However, if using the turn-level encoder, this pair's input sequence is encoded and the turn-level state is updated. During evaluation on the development and test sets, we limit generation to 300 tokens.

If not using the turn-level encoder, we delimit the previous and current utterances with a special delimiter token when encoding the inputs (Section 4.2). The corresponding encoder hidden states of the delimiters are not used during attention.

Parameters We use word embeddings of size 400. The word embeddings are not pre-trained. Utterance age embeddings are of size 50. Query segment age embeddings are of size 64. For all models that use query segment copying, $g = 4$. All LSTMs have a hidden size of 800. The sizes of the learned matrices are: $\mathbf{W}^{\mathbf{A}} \in \mathbb{R}^{850\times 800}$, $\mathbf{W}^{m} \in \mathbb{R}^{1650\times 800}$, $\mathbf{W}^{o} \in \mathbb{R}^{800\times |V^o|}$, $\mathbf{b}^{o}_{w} \in \mathbb{R}^{|V^o|}$, and $\mathbf{W}^{S} \in \mathbb{R}^{800\times 1600}$. Unless otherwise noted, the initial hidden state and cell memory of all LSTMs are zero-vectors. All parameters are initialized randomly from a uniform distribution $U[-0.1, 0.1]$.

D Results

D.1 Overfitting in Original Data Split

We assess overfitting on the training set of the original split of the data by measuring how performance changes when data pre-processing is removed. Table 4 shows that removing data pre-processing lowers the performance of FULL by around 9% when using our data split. However, on the original split of the data, performance drops by only 1.3%. This relatively high performance is only possible due to learned biases within the scenarios.

Split	Model	Strict Denotation
Original	FULL	68.4±0.2
	– preprocess.	67.1±1.0 **(-1.3)**
Ours	FULL	62.5±0.9
	– preprocess.	53.0±8.5 **(-9.5)**

Table 4: Strict table accuracy results using our model with and without pre-processing on the development sets of the original data split and our data split.

Model	**Query**	**Denotation**	
		Relaxed	**Strict**
FULL-GOLD	42.1±0.8	66.6±0.7	66.1±0.7
– turn-level encoder	42.5±1.7	66.3±1.7	65.7±1.9
– batch re-weighting	41.1±0.7	65.5±0.3	64.8±0.6
– input pos. embs.	38.0±0.4	61.4±1.1	60.5±1.1
– anon. scoring	40.8±0.7	64.7±1.4	63.9±1.3
– pre-processing	35.3±6.9	57.9±8.4	57.4±8.2

Table 5: Ablations on FULL-GOLD, showing performance on the development set averaged over all utterances. Gold queries are provided for previous query segment extraction. In each model, $h = 3$. We show average and standard deviation over five trials for each model.

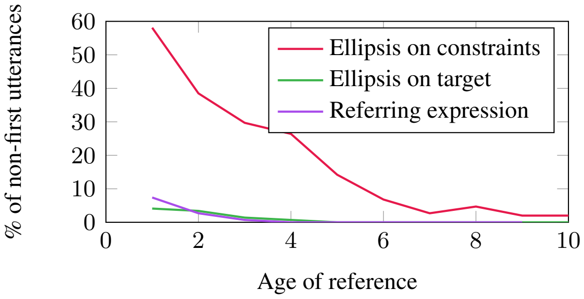


Figure 6: Distribution of referent ages over sampled utterances in the development set.

D.2 Ablations using Gold Previous Queries

Table 5 shows results on ablating components from FULL when extracting segments from previous gold queries.

E Analysis

E.1 Data Analysis

The two most common contextual phenomena in ATIS are ellipsis and referring expressions. Ellipsis refers to utterances that omit relevant information mentioned in the past. For example, $\bar{y}_2$ in Figure 2 contains flight endpoint constraints that are not present in $\bar{x}_2$. To recover these constraints, the model must resolve an implicit reference to $\bar{x}_1$. Referring expressions explicitly refer to earlier constraints or system responses. In the example utterances, the phrase *which ones* in $\bar{x}_3$ refers to the table returned by executing $\bar{y}_2$.

An analysis of twenty development-set interactions shows that utterances after the first turn contain on average 1.85 omitted constraints, and 60.1% of utterances after the first turn omit at least one constraint. Figure 6 shows the distribution of referent ages over all utterances in the twenty analyzed interactions from the development set, considering three major types of references: ellipsis on constraints, ellipsis on the attributes targeted by the user request (e.g., if a user omits the target attribute in the current utterance, but it is clear from context), and referring expressions (explicit mentions of previous constraints). 9% of utterances after the first utterance omit the target attribute; 11% contain a referring expression to previous results or constraints.

E.2 Attention Analysis

Figures 7, 8, and 9 demonstrate the robustness of the full approach when attending over multiple previous utterances. Each figure shows attention while processing the same example, which is the third utterance in an interaction from the development set. This interaction exemplifies a user focus state change. To process the third utterance, *show flights*, correctly, the model must be able to recover constraints mentioned in the first utterance, although the user briefly changes focus in the second utterance. In each figure, the query is shown on the lefthand side, and the utterances are at the bottom. The opacity of each cell represents the attention probability for each token during each generation step. Darker lines in the figure separate the three utterances.

Figure 7 shows the attention computed by FULL when provided with gold query segments. The decoder attends over entities in the previous utterance, including the flight endpoints and date, when generating the query. Figure 8 shows the attention computed by FULL-0 when provided with gold query segments. This model does not have explicit access to the constraints mentioned in the first utterance. It is unable to recover these constraints, and instead makes up constraints, such as endpoints and a date, while also making a semantic mistake, `city.city_name = 1200`. This demonstrates the robustness provided by the attention mechanism in the full model. For comparison, Figure 9 shows the attention computed by S2S+ANON on the same example. Like FULL, this model attends over entities in the first utterance, including flight endpoints and the date. Both FULL and S2S+ANON were able to recover the correct query.

Figures 10, 11, and 12 demonstrate how the ability to copy query segments is critical to our model's performance. FULL and FULL-0 are able to recover the correct query. In both cases, `SEGMENT_9`, which is extracted from the previous query, contains the flight endpoint constraints (from the first utterance), as well as a constraint that the flight be the shortest one available (from the second utterance). S2S+ANON is unable to recover the minimum-time constraint, even though it has the ability to attend over the relevant tokens in the second utterance. The ability of FULL-0 to recover this constraint without attending on previous utterances demonstrates the benefit that copying previous segments provides. These examples also show that when copying query segments, fewer decoding steps are required.

E.3 Contextual Analysis

We construct several example interactions targeting the contextual phenomena discussed in Section 9, and test FULL against them.

$\bar{x}_1$: show me flights from seattle to denver after 6am
$\bar{x}_2$: leaving after 7am
$\bar{x}_3$: leaving after 8am
$\bar{x}_4$: leaving before 9am
$\bar{x}_5$: leaving after 10am
$\bar{x}_6$: leaving from san francisco

This example shows ellipsis of both constraints (flight endpoints) and target attribute (flights) while modifying existing constraints. FULL is able to predict correct queries for all new utterances as the user continues to change constraints and elided values increase in age. Whether $\bar{y}_6$ includes a time constraint or not is ambiguous; FULL generates a query to search for all flights from San Francisco to Denver regardless of time.

$\bar{x}_1$: show me flights from seattle to denver
$\bar{x}_2$: leaving after 7am
$\bar{x}_3$: stopping in san francisco
$\bar{x}_4$: on american airlines
$\bar{x}_5$: which is the cheapest
$\bar{x}_6$: with breakfast

This example shows ellipsis of both constraints and target attribute while adding constraints. FULL is able to predict correct queries for all utterances in this interaction.

$\bar{x}_1$: show me flights from seattle to denver after 6am
$\bar{x}_2$: how much does it cost
$\bar{x}_3$: what meal is offered
$\bar{x}_4$: which airlines are available
$\bar{x}_5$: what type of airplane does it use
$\bar{x}_6$: what ground transportation is available

This example shows ellipsis on constraints (endpoints and time) while changing the target attribute. Additionally, it demonstrates change in focus while the user switches from asking for flights to asking about airlines. While ambiguous, the final utterance $\bar{x}_6$ is interpreted as finding ground transportation in Denver. FULL is able to recover correct queries for all utterances in this interaction.

$\bar{x}_1$: show me flights from seattle to denver after 6am
$\bar{x}_2$: what ground transportation is available in denver
$\bar{x}_3$: i want to fly on delta

This example shows ellipsis when user focus changes, temporarily rendering the origin city and time constraints irrelevant. FULL predicts queries for the first two utterances correctly, but fails to generate the correct origin city constraint in the third prediction. Instead, it generates a constraint that Denver is the origin city. When $\bar{x}_2$ is removed from the interaction, both predictions are correct.

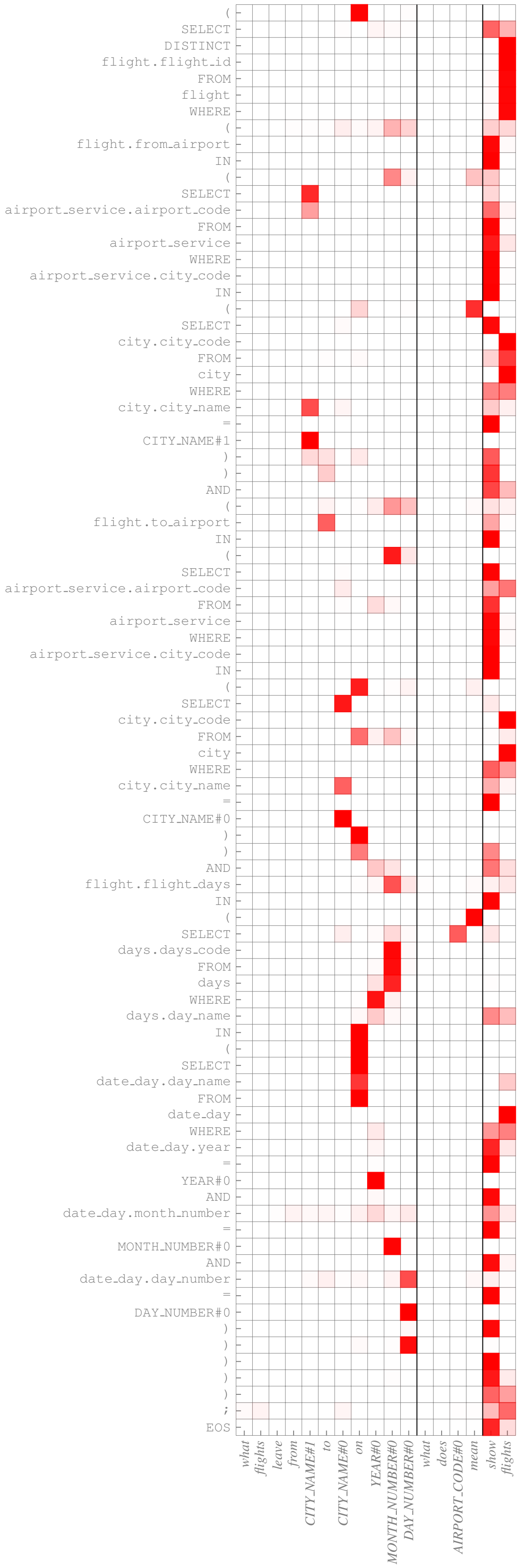


Figure 7: Attention computed by FULL after a user state focus change. Compare to Figures 8 and 9.

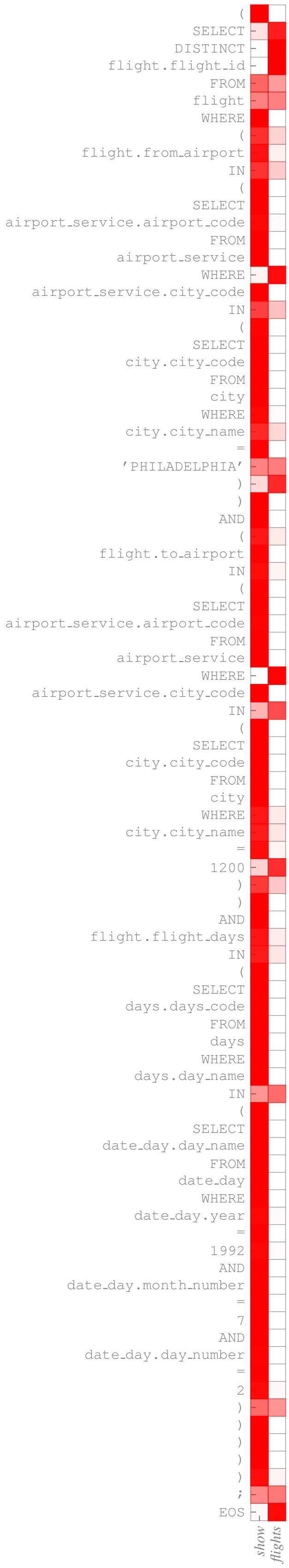


Figure 8: Attention computed by FULL-0 when generating a query for an utterance after a user state focus change. The model does not have explicit access to flight endpoint or date constraints, and is unable to generate the correct query. Figure 7 shows FULL on this example; Figure 9 shows S2S-ANON.

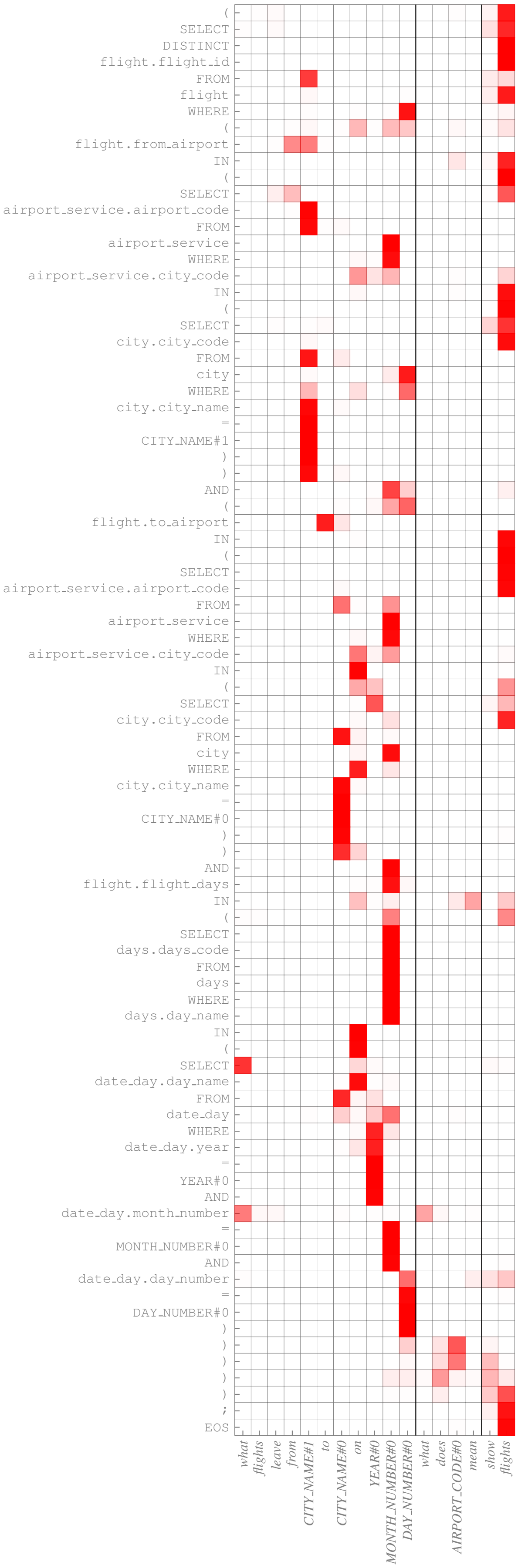


Figure 9: Attention computed by S2S+ANON after a user state focus change. Compare to Figures 7 and 8.

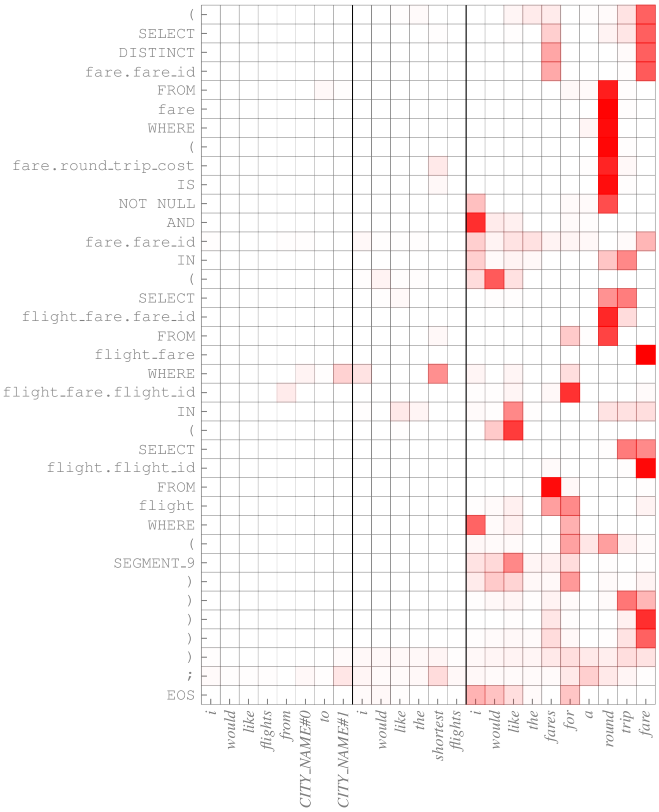


Figure 10: Attention computed by FULL, demonstrating copying of query segments. Figure 11 shows FULL-0 on this example; Figure 12 shows S2S-ANON.

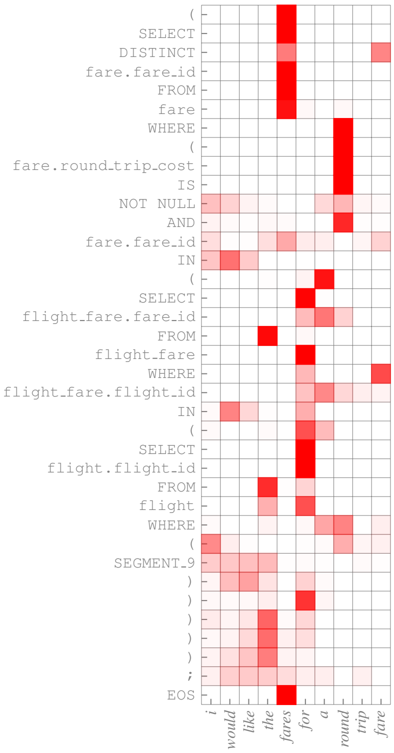


Figure 11: Attention computed by FULL-0 that makes use of query segments to recover constraints it does not otherwise have explicit access to. Figure 10 shows FULL on this example; Figure 12 shows S2S-ANON.

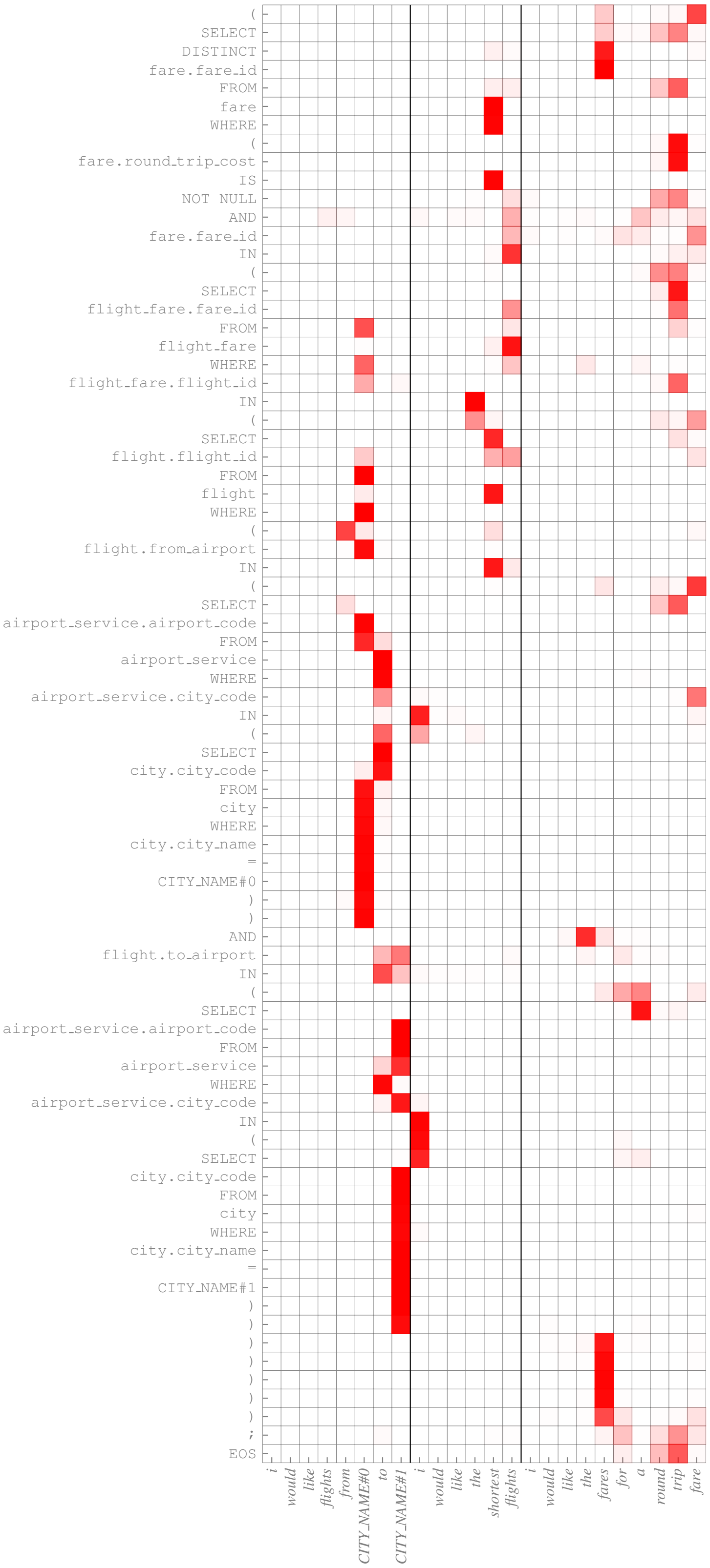


Figure 12: Attention computed by S2S-ANON, demonstrating a failure in recovering constraints. Figure 10 shows FULL on this example; Figure 11 shows FULL-0.