SpeakNav: Voice-based Route Description Language Understanding for Template-driven Path Search

Bolong Zheng[1], Lei Bi[1], Juan Cao[1], Hua Chai[2], Jun Fang[2], Lu Chen[3], Yunjun Gao[3]
Xiaofang Zhou[4], Christian S. Jensen[5]
[1]Huazhong University of Science and Technology, [2]Didi Chuxing, [3]Zhejiang University,
[4]Hong Kong University of Science and Technology, [5]Aalborg University
bolongzheng@hust.edu.cn,leibi@hust.edu.cn,juancao@hust.edu.cn,chaihua@didiglobal.com
fangjun@didiglobal.com,luchen@zju.edu.cn,gaoyj@zju.edu.cn,zxf@cse.ust.hk,csj@cs.aau.dk

ABSTRACT

Many navigation applications take natural language speech as input, which avoids users typing in words and thus improves traffic safety. However, navigation applications often fail to understand a user's free-form description of a route. In addition, they only support input of a specific source or destination, which does not enable users to specify additional route requirements. We propose a SpeakNav framework that enables users to describe intended routes via speech and then recommends appropriate routes. Specifically, we propose a novel Route Template based Bidirectional Encoder Representation from Transformers (RT-BERT) model that supports the understanding of natural language route descriptions. The model enables extraction of information of intended POI keywords and related distances. Then we formalize a template-driven path query that uses the extracted information. To enable efficient query processing, we develop a hybrid label index for computing network distances between POIs, and we propose a branch-and-bound algorithm along with a pivot reverse B-tree (PB-tree) index. Experiments with real and synthetic data indicate that RT-BERT offers high accuracy and that the proposed algorithm is capable of outperforming baseline algorithms.

PVLDB Reference Format:
Bolong Zheng, Lei Bi, Juan Cao, Hua Chai, Jun Fang, Lu Chen, Yunjun Gao, Xiaofang Zhou, Christian S. Jensen. SpeakNav: Voice-based Route Description Language Understanding for Template-driven Path Search. PVLDB, 14(12): 3056 - 3068, 2021.
doi:10.14778/3476311.3476383

1 INTRODUCTION

Due to the prevalence of GPS-equipped devices, navigation applications are used frequently by people as they travel as part of their daily lives. Traditional navigation applications use a keyboard or a touch pad to obtain input, requiring users to take their eyes off the road and hands off the wheel, which can be dangerous when driving. In contrast, voice-based human-machine interaction enables drivers to better focus on driving, thus decreasing traffic accidents. Over the past decades, mature speech recognition engines have been developed that convert speech into text. However, it remains challenging for existing voice-based navigation systems to understand free-form natural language descriptions of desired route properties. Thus, users still need to provide speech input according to designated sentence patterns; otherwise, existing systems fail.

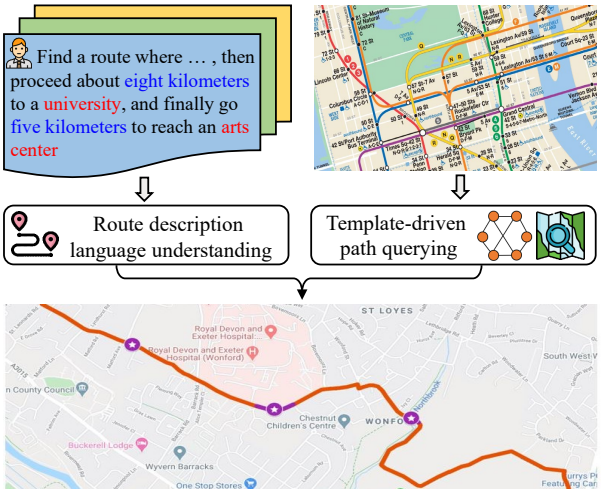


Figure 1: The Components of SpeakNav

The increasing availability of point of interest (POI) data provides an opportunity to enable users to describe locations when retrieving desirable routes using navigation applications. In contrast, existing applications typically support only input of a single source or destination and do not support the specification of additional desired route properties. We aim to support use cases such as the following: a user wants to retrieve the route to a restaurant with a theater within walking distance, so that the user can watch a show after dinner. We also want to support use cases where a user has only partial directions to a destination. For example, a user may wish to issue a query like the following: "Find a route where I go straight for about fifteen kilometers passing by a restaurant, then proceed about eight kilometers to a university, and finally go five kilometers to reach an arts center". In both use cases, the query results are routes that pass by different types of POIs within desired distances. Existing navigation systems fail to support such queries.

To extend the use cases supported by navigation systems, we propose the SpeakNav framework, shown in Fig. 1. SpeakNav provides two innovations: route description language understanding

Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476383

and template-driven path querying. To the best of our knowledge, SpeakNav is the first proposal that offers combined support for these innovations. SpeakNav thus addresses two challenges:

- **Route description language understanding.** Users have their own preferred natural language formulations of route properties in terms of POIs and distances. Taking such descriptions, with different sentence patterns, as input, the proposed model must determine the intended order of specified POIs along the route. Specifically, the model needs to capture accurately two relationships. The first is the relation between POIs and distances, and the second is the order relationship between POIs.
- **Template-driven path querying.** We consider a template for route descriptions that includes POI keywords and distances between POIs. However, no direct topological relationships among the POIs in the road network are specified. Thus, POIs must be mapped to the road network to measure the network distances between them. Furthermore, road networks come with large numbers of POIs, each of which is described by multiple keywords. As candidate paths are supposed to cover all POI keywords given by a user, the number of candidate paths increases exponentially with the number of query keywords, making it challenging to find desired routes efficiently.

In SpeakNav, we propose a joint intent classification and slot filling model based on Bidirectional Encoder Representation from Transformers (BERT) [7], which we call Route Template based BERT (RT-BERT), to extract POI keywords and to distinguish the order of them. In addition, we introduce an intent-attention mechanism to model the dependencies between intent labels and slot words, and we introduce a self-attention mechanism to strengthen the order relations between POIs. In order to find POIs with accompanying distances, we first propose a hybrid label index to measure the distances between POIs. To enable efficient route retrieval, we construct a pivot reverse B-tree (PB-tree) that preserves POI keyword information and network distances. We then develop a branch-and-bound (BAB) algorithm that exploits the PB-tree to prune non-qualifying POIs.

The major contributions are summarized as follows.

- We propose a route template based BERT model, RT-BERT, with an encoder-decoder framework for joint intent classification and slot filling.
- We construct a hybrid label index and a PB-tree index to prune POIs. For the template driven path querying, we develop a branch-and-bound algorithm to enable efficient path retrieval.
- We generate the first annotated text dataset for the task of Route Description Natural Language understanding, called RDNL, which contains 4,900 sentences with different sentence patterns.
- We conduct an extensive performance study on real datasets and the generated dataset, which indicates that RT-BERT is capable of outperforming the state-of-art baselines in terms of accuracy, and that the BAB algorithm is capable of outperforming the baselines in terms of efficiency.

The remainder of the paper is organized as follows. Related work is covered in Section 2. We formulate the problem in Section 3. Section 4 introduces our solution. Section 5 provides the experimental results. Finally, Section 6 concludes the paper.

2 RELATED WORK

2.1 Natural Language Understanding

Natural language understanding in our setting involves two tasks: intent classification and slot filling. Early studies consider these two tasks separately. Intent classification aims to determine the semantics of a sentence and check if it is navigation-related. Therefore, the popular classifiers can be easily applied, such as support vector machines [12] and deep neural networks [16, 21]. Slot filling can be viewed as a sequence labeling task that maps an input word sequence to the corresponding slot label sequence. Yao et al. [29] present an adaptation of Recurrent Neural Network (RNN) to predict the slot label, which outperforms traditional and previous neural network based approaches. Unlike these works, we model dependencies between the intent labels and slot tags to determine the order relationship based on semantics.

Recent studies that consider the intent classification and slot filling tasks jointly have achieved great progress. Goo et al. [10] focus on learning explicit slot-intent relations by introducing a slot-gated mechanism into an attention model, which improves the performance of slot filling on the learned intent result. E et al. [8] introduce an SF-ID network that establishes direct connections between intent classification and slot filling, allowing the tasks to promote each other. In addition, multi-task deep learning architectures [11, 13] enable reinforcement among domain determination, intent classification, and slot filling. However, these methods have not been applied to navigation, and no large-scale human-labeled training datasets exist, which results in poor generalization capabilities, especially for rare words. To address this problem, the state-of-the-art models utilize pre-trained language models. Chen et al. [5] propose a joint intent classification and slot filling model based on the Bidirectional Encoder Representation from Transformers (BERT) [7], which pre-trains deep bidirectional representations using large-scale unlabeled texts. In this setting, we introduce multiple attention mechanisms to enhance the dependencies between intent labels and slot tags.

2.2 Keyword-aware Route Search

Keyword-aware route search has received considerable attention. Cao et al. [4] study a keyword-aware optimal route query that finds an optimal route that covers a set of keywords and satisfies a budget constraint. Yao et al. [28] study a multi-approximate-keyword routing (MAKR) problem that takes a source, a destination, and a set of keyword-threshold pairs as input. The query returns the shortest route that covers at least one matching POI per keyword with similarity above the corresponding threshold. These algorithms do not associate distance values with the query keywords. Zhao et al. [30] study a group-based keyword-aware route (GKAR) query problem that aims to find a route that passes by a sequence of POIs that match a sequence of query keywords while minimizing the route cost. However, our problem is different from these problems so that their algorithms are not applicable. We introduce the notion

Table 1: Summary of Notation

Notation	Definition
x	A route description text sequence
y	A intent label
y^s	A slot tag sequence
H	Contextual semantic representation embedding obtained by BERT
H^I	Hidden state vector after intent-attention mechanism
H^S	Hidden state vector after self-attention mechanism
H'	The concatenation of H and H^S
H^B	Hidden state vector of BiLSTM
$G = (V, E, W)$	A road network with vertices V, edges E, and weights W
$p = (l, \Phi)$	A POI with location $p.l$ and a keyword set $p.\Phi$
$t(w, d)$	A template with keyword w and distance d
$\mathcal{P}(p_0.l, \ldots, p_k.l)$	A path from $p_0.l$ to $p_k.l$
$d_G(l_i, l_j)$	Network distance between l_i and l_j
$L(v)$	Vertex label of v
$L_p(p)$	POI label of p

of route template distance, which quantifies the degree of matching of a path to a query. As a result, the paths returned by our proposed algorithm are able to match a user's query more accurately. Zheng et al. [31] investigate so-called clue-based route search (CRS) that allows users to provide clues on the textual and spatial context of a route. While related closely to our problem, they simply map POIs to network vertices and do not consider the distances between them, which yields reduced accuracy. In addition, the templates in our work are extracted from natural language, and the distances in templates can be set to default values, which is not supported by Zheng et al. [31].

3 PRELIMINARIES

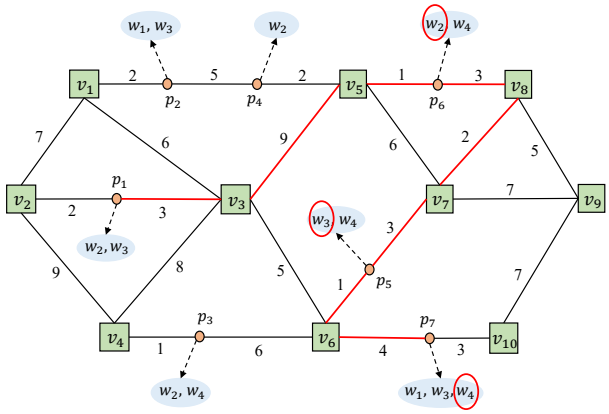
SpeakNav solves two problems: (1) route description language understanding and (2) template-driven path querying. We introduce relevant background knowledge and formulate the problems. Frequently used notation is summarized in Table 1.

3.1 Route Description Language Understanding

The objective of the route description language understanding is to extract template information for the expected route, i.e., POI keywords and corresponding distances, from the speech input. It consists of two tasks: intent classification and slot filling.

Definition 1 (Intent Classification). *The intent classification is a binary text classification problem. Given a path description text sequence $x = (x_1, x_2, \ldots, x_n)$, intent classification generates a tag $y \in \{0, 1\}$ for each x indicating whether x is a path description text sequence.*

Definition 2 (Slot Filling). *Slot filling identifies template information from a path description. Slot filling annotates a path description $x = (x_1, x_2, \ldots, x_n)$ by slot tags $y^s = (y_1^s, y_2^s, \ldots, y_n^s)$, where slot tag y_i^s corresponds to word x_i and represents a POI keyword or a distance value, or indicates that x_i is irrelevant.*



w_1: theater; w_2: restaurant; w_3: university; w_4: arts center.

Figure 2: Running Road Network Example

3.2 Template-driven Path Querying

A template-driven path querying aims to find an optimal path based on template information retrieved from input speech.

Definition 3 (POI). *A point of interest (POI) $p_i = (l, \Phi)$ has a location $p_i.l$ and a set of keywords $p_i.\Phi$.*

For example in Fig. 2, we have a set of POIs $P = \{p_1, p_2, \ldots p_7\}$ with $p_1.\Phi = \{w_2, w_3\}$, and $p_7.\Phi = \{w_1, w_3, w_4\}$.

Definition 4 (Road Network). *A road network is modeled as a weighted, undirected graph $G = (V, E, W)$, where V is a vertex set, E is an edge set, $W : E \to \mathbb{R}$ assigns positive weights to all edges that capture the edge lengths.*

For any two locations l_1 and l_2, the network distance in G between l_1 and l_2, denoted as $d_G(l_1, l_2)$, is the length of the shortest path between them. For example, we have $d_G(v_1, v_5) = 9$, $d_G(p_1.l, p_6.l) = 13$, and $d_G(v_1, p_1.l) = 9$.

Definition 5 (Template). *We define a template as $t(w, d)$, where w is a keyword and d is a distance.*

We use templates to specify types of POIs and distances that characterize desirable routes. For example, the template t("restaurant", 15 km) indicates that a route should pass by a POI with keyword "restaurant" at about 15 kms into the route. Note that, the templates correspond to the slot tags obtained from slot filling.

Definition 6 (Candidate path). *A template-driven path query (TPQ) is given by $Q = (l_0, \mathcal{T})$, where l_0 is the current user location, and $\mathcal{T} = \langle t_1(w_1, d_1), t_2(w_2, d_2), \ldots, t_k(w_k, d_k)\rangle$ is a template sequence. We consider a path $\mathcal{P}(p_0.l, p_1.l, \ldots, p_k.l)$ as a result candidate if it satisfies the following conditions.*

(1) For the POI sequence $\langle p_0, p_1, \ldots, p_k\rangle$, where $p_0.l = l_0$, we have $w_i \in p_i.\Phi$, $1 \le i \le k$.
(2) For $1 \le i \le k$, subpath $\mathcal{P}(p_{i-1}.l, \ldots, p_i.l)$ is a shortest path.

Definition 7 (Route Template Distance). *In order to quantify the degree of matching of a subpath $\mathcal{P}(p_{i-1}.l, \ldots, p_i.l)$ to a template $t_i(w_i, d_i)$, the route template distance is computed as follows.*

$$d_R^i = \frac{|d_G(p_{i-1}.l, p_i.l) - d_i|}{\epsilon \cdot d_i}, \tag{1}$$

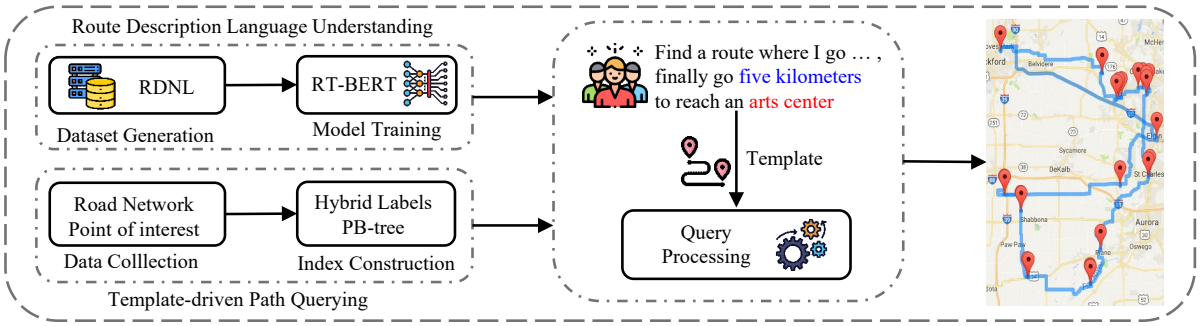


Figure 3: SpeakNav Framework Overview

where ϵ is a tolerance parameter that relaxes the network distance threshold between p_i and p_{i-1} to the interval $[d_i(1-\epsilon), d_i(1+\epsilon)]$. Here, any increasing function that normalizes the route template distance to $[0,1]$ can be applied.

The overall route template distance between a candidate path $\mathcal{P}$ and a query Q is computed as follows.

$$d_R(\mathcal{P}, Q) = \max_{1 \le i \le k} d_R^i \tag{2}$$

The motivation for using Eq. 2 is that the maximum route template distance of all templates naturally controls the overall quality of the candidate path. It is a widely adopted method for the problem of optimizing an objective score with multiple components.

Definition 8 (Template-driven path querying (TPQ)). *A template driven path query (TPQ) $Q = (l_0, \mathcal{T})$ returns the candidate path with the smallest route template distance to the query.*

Specifically, if there exists a template with no distance specified in a query, the goal is to minimize the length of the candidate path.

Example 1. *Consider the speech input "Find a route where I go straight for about fifteen kilometers passing by a restaurant, then proceed about eight kilometers to a university, and finally go five kilometers to reach an arts center?" We first perform intent classification, and then fill slot tags to obtain the following template sequence: $\mathcal{T} = \langle t_1(\text{"restaurant"}, 15), t_2(\text{"university"}, 8), t_3(\text{"arts center"}, 5)\rangle$. The words "restaurant", "university" and "arts center" correspond to the keywords "w_2", "w_3" and "w_4" in Fig. 2.*

Assume that the user's current location $l_0 = p_1.l$, and ϵ is set to 0.4. Therefore, we have $Q = (p_1.l, \mathcal{T})$. The paths $\mathcal{P}_1(p_1.l, p_6.l, p_5.l, p_7.l)$ and $\mathcal{P}_2(p_1.l, p_3.l, p_7.l, p_5.l)$ are candidates with $d_R(\mathcal{P}_1, Q) = 0.333$ and $d_R(\mathcal{P}_2, Q) = 0.625$. Therefore, $\mathcal{P}_1$ is returned as the result.

3.3 Framework Overview

The SpeakNav framework, as shown in Fig. 3, consists of two components, route description language understanding and template-driven path querying. A demo is described elsewhere [3].

SpeakNav integrates the speech recognition engine iFLYTEK[1] to convert the speech to a text sequence. Then, we propose a route template based BERT model (RT-BERT) that uses an encode-decode framework to extract the template information in the text sequence. The encoder generates feature vectors based on the pre-trained BERT model, which contain contextual semantics for the text sequence. The decoder utilizes the vector features to generate intent labels and slot tags. The intent labels indicate whether the text sequence is a template based route description. The slot tags capture that the words in a path description belong to the POI keywords, are distance values, or are irrelevant words.

To enable efficient template-driven path query processing, we develop a hybrid label index and a PB-tree index in offline preprocessing phase. The hybrid label index computes network distances between POIs efficiently. The PB-tree is used to accelerate the query processing of a branch-and-bound algorithm.

[1] https://www.xfyun.cn/

4 PROPOSED SOLUTION

We proceed to solve the two sub-problems, route description language understanding and template-driven path querying.

4.1 Route Template-based BERT

We propose a route description language understanding model, called Route Template-based BERT (RT-BERT), based on an encoder-decoder framework that enables joint intent classification and slot filling. The structure of RT-BERT is shown in Fig. 4. Since the pre-trained language model has significantly improved the performance of natural language processing tasks, we use the Bidirectional Encoder Representation from Transformer (BERT) [7] as the encoder to generate pre-trained word vectors, while utilizing a Bidirectional Long Short-Term Memory (BiLSTM) model [23] and a Conditional Random Field (CRF) model [22] as the decoder. In order to capture the different effects of intent labels on slot words and non-slot words, we introduce an intent-attention mechanism. RT-BERT also models the order relations between the user intended keywords in the input text sequence using a self-attention mechanism.

4.1.1 ***Encoder.*** To build a general model, we utilize BERT as the encoder since it is pre-trained on large unlabeled text collections with multiple language training tasks. BERT is composed of a stack of multi-layer bidirectional Transformer encoders [24], which perform well in parallel computing settings and capture contextual semantics. In order to learn the sequentiality and the semantic similarity of sentences, we concatenate positional embeddings and segment embeddings with WordPiece embeddings [27] in the input representation.

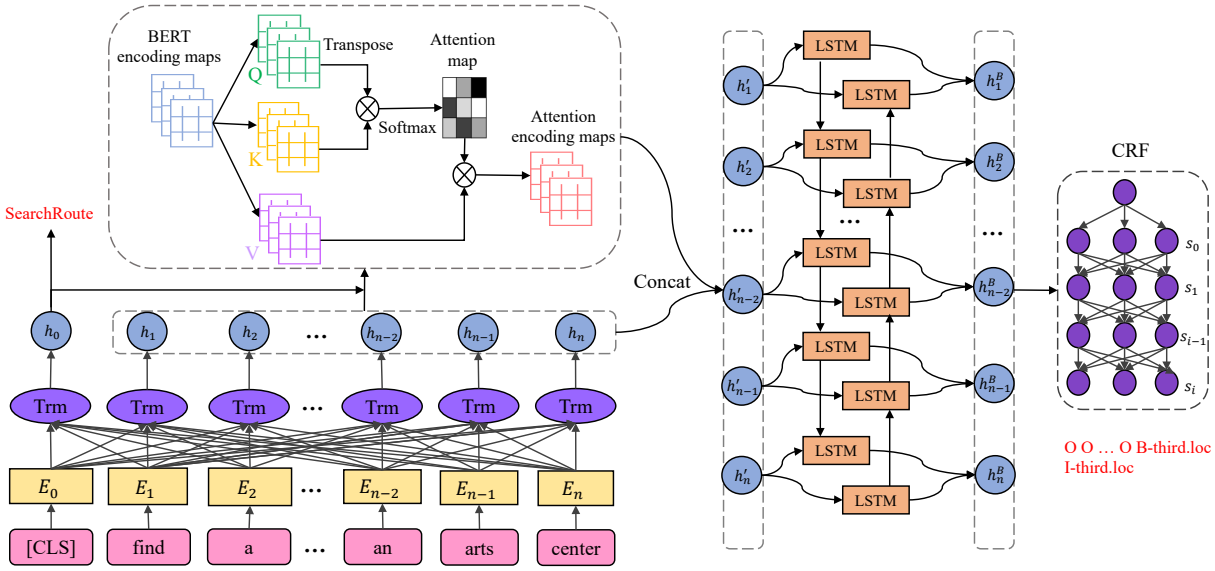


Figure 4: Structure of RT-BERT

For intent classification, a special classification token [CLS] is inserted before the input text sequence. The hidden state of the token [CLS] can be used as the semantic representation of the whole sentence, thus determining the intent. Specifically, given an input text sequence $x = (x_1, \ldots, x_n)$, we denote the token [CLS] as x_0 and concatenate it with the input sequence to obtain $(x_0, x_1, x_2, \ldots, x_n)$. The encoder generates a pre-trained vector:

$$H = \text{BERT}(x_0, x_1, x_2, \ldots, x_n), \tag{3}$$

where $H = (h_0, h_1, h_2, \ldots, h_n) \in \mathbb{R}^{(n+1)\times\lambda}$, and λ is the dimensionality of the hidden state of a word.

The element h_0 is the hidden state of the special token ([CLS]) that is fed directly into a fully connected layer and a softmax layer to generate the intent label as follows.

$$y = \text{softmax}(Wh_0 + b), \tag{4}$$

where W is a weight matrix and b is a bias term.

Each element h_i $(1 \le i \le n)$ is the contextualized representation of input word x_i for slot filling. In addition, we introduce multiple attention mechanisms, BiLSTM, and CRF to decode the sequence and fill the slot tags.

4.1.2 ***Attention Mechanism.*** Attention mechanisms have been used widely for their support for parallel computation and dependency modeling capabilities [9, 26]. Attention mechanisms make it possible to focus more on important features and less on unimportant ones. Therefore, we incorporate intent-attention and self-attention mechanisms into the framework to model dependencies among intent labels and slot tags, and to model the order relations between user intended keywords. In particular, a scaled dot-product attention is used to compute attention scores as follows.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^\top}{\sqrt{\lambda_K}})V, \tag{5}$$

where Q (query) is the attention weight matrix that represents the correlation between words, K (key) is the weight index, and V (value) is the word vector at the current training time step. Eq. 5 is equivalent to normalizing the attention distribution of all word pairs in a sentence and attaching it to the word vector. Parameter λ_K represents the dimensionality of K and is used to prevent the gradient from disappearing.

Intent-attention. We observe that the intent contextual information is more important for slot words than for non-slot words. For example, the input text sequence "Find a route ... an arts center" in Fig. 4 is a route description. Its intent label is set to 'SearchRoute' that is determined by the hidden state of the token [CLS]. The intent label is supposed to have larger relevance to the slot words 'arts' and 'center' than the non-slot words 'find', 'a', etc. The intent-attention mechanism is used for this purpose. We remove h_0 from H to form $H_1 = (h_1, h_2, \ldots, h_n) \in \mathbb{R}^{n\times\lambda}$, where h_0 is the hidden state of [CLS] characterizing the intent label to some extent. In the intent-attention mechanism, we have $Q = H_0 = (h_0, h_0, \ldots, h_0) \in \mathbb{R}^{n\times\lambda}, K = V = H_1$. The attention score α_i between h_0 and the hidden state h_i is computed as follows.

$$\alpha_i = \text{softmax}(\frac{h_0 \cdot h_i^\top}{\sqrt{\lambda}}) \tag{6}$$

Then, we compute the output vector h_i^I of the intent-attention mechanism according to the α_i as follows.

$$h_i^I = \sum_{i=1}^{n} \alpha_i h_i \tag{7}$$

Self-attention. Following the intent-attention layer, we apply a self-attention mechanism. This mechanism enables the computation of attention weights between each pair of tokens in a single sequence, so it works well when capturing long-range dependencies. For a template-based route description text sequence, we use

the self-attention mechanism to capture dependencies between keywords that are far apart in a sentence. In the self-attention mechanism, we have $Q = K = V = H^I$. The output vector h_i^S is computed as follows.

$$h_i^S = \sum_{i=1}^{n} \text{softmax}(\frac{h_i^I \cdot (h_i^I)^\top}{\sqrt{\lambda}}) \cdot h_i^I \tag{8}$$

To retain the powerful context-dependent sentence representation provided by the RT-BERT encoder, we concatenate h_i^S after the attention mechanism with h_i, that is, $h_i' = [h_i, h_i^S]$. The decoder is initiated with the final state $H' = (h_1', h_2', \ldots, h_n')$.

4.1.3 ***Slot Filling Decoder.*** The decoder aims to output a slot tag sequence by decoding the input vector H'. To capture long-term dependencies of POIs and to make use of both forward and backward features, we decode $H' = (h_1', h_2', \ldots, h_n')$ using a Bidirectional Long Short-Term Memory (BiLSTM) network that contains a forward LSTM [15] and a backward LSTM. The BiLSTM reads the input sequence H' in its original and reverse orders and generates two hidden states in each time step. The final hidden state h_τ^B at time step τ is the concatenation of the forward state $\overrightarrow{h_\tau}$ and backward state $\overleftarrow{h_\tau}$:

$$\begin{aligned} \overrightarrow{h_\tau} &= \overrightarrow{\text{LSTM}}(h_\tau', \overrightarrow{h_{\tau-1}}) \\ \overleftarrow{h_\tau} &= \overleftarrow{\text{LSTM}}(h_\tau', \overleftarrow{h_{\tau+1}}) \\ h_\tau^B &= [\overrightarrow{h_\tau}, \overleftarrow{h_\tau}] \in \mathbb{R}^m \end{aligned} \tag{9}$$

Hence, we obtain a complete sequence of all hidden states: $H^B = (h_1^B, h_2^B, \ldots, h_n^B) \in \mathbb{R}^{n \times m}$, where m is the dimensionality of a hidden state in BiLSTM. A linear layer is used to map each hidden state vector in H^B from m dimensions to z dimensions, where z is the number of unique slot tags, and each slot tag is indexed in the range $[1, z]$. Note that, the set of slot tags is obtained when generating the dataset (See Section 5.1.1). As a result, the output of BiLSTM is a matrix $E = (E_1, E_2, \ldots, E_n) \in \mathbb{R}^{n \times z}$, where $E_{i,j}$ is the probability that the slot tag of the i-th word is the j-th slot tag.

However, BiLSTM fails to capture dependencies between adjacent slot tags. In Example 1, "arts center" is the third keyword of the user's desired route, so the slot tags of "arts" and "center" should belong to a same category. It is incorrect that the slot tag of "arts" represents one keyword and the slot tag of "center" represents another keyword. Therefore, we add a CRF layer after BiLSTM that takes advantage of the sequence-level label information offered by CRF to obtain a better prediction sequence. The parameter of CRF is a transition matrix $A \in \mathbb{R}^{(z+2)\times(z+2)}$ that can be learned automatically by the CRF layer during training, and $A_{i,j}$ denotes the transition score from the i-th slot tag to the j-th slot tag. For a slot tag sequence $y^s = (y_1^s, y_2^s, \ldots, y_n^s)$ of a sentence $x = (x_1, x_2, \ldots, x_n)$, the score function is computed as follows.

$$s(x, y^s) = \sum_{i=0}^{n} A_{\mu_i, \mu_{i+1}} + \sum_{i=1}^{n} E_{i, \mu_i} \tag{10}$$

where μ_i and μ_{i+1} are the corresponding indexes of slot tags y_i^s and y_{i+1}^s. The score of the whole sequence equals the sum of the scores of all words in the sentence, which is determined by the output matrix E of the BiLSTM and the transition matrix A of CRF. Given a sentence x and its ground truth y^x, the goal is to maximize the score of y^x in all possible prediction sequences. The model is trained using the negative log-likelihood loss function for backpropagation, which is formalized as follows.

$$\begin{aligned} -\log(\Pr(y^x|x)) &= -\log(\frac{e^{s(x,y^x)}}{\sum_{y' \in Y} e^{s(x,y')}}) \\ &= -s(x, y^x) + \log(\sum_{y' \in Y} e^{s(x,y')}), \end{aligned} \tag{11}$$

where Y represents the set of all possible prediction sequences of x. We obtain the optimal sequence y^* from Y by using the Viterbi algorithm [25], and the keywords and distances determined by y^* are then used in a template-driven path query.

Example 2. *For the text sequence "Find a route where I go straight for about ..." in Example 1, we first determine that the intent label is "SearchRoute", which is related to template-driven path search. Next, we generate slot tags as follows: "O O O O O O O O O B-first.dis I-first.dis O O O B-first.loc O O O B-second.dis I-second.dis O O B-second.loc O O O B-third.dis I-third.dis O O O B-third.loc I-third.loc", which is annotated by Begin-Inside-Outside (BIO, details are presented in Section 5.1.1).*

4.2 Template Driven Path Query Processing

The template driven path query (TPQ) returns the path that has the smallest route template distance to the query extracted by the RT-BERT model. First, we propose a hybrid label index that enables efficient network distance computation between POIs. Next, we develop a pivot reverse B-tree (PB-tree) on top of the hybrid label index to locate POIs when finding the next matching POI. Using these index structures, we design a branch-and-bound (BAB) algorithm that uses the filter-and-refinement paradigm.

4.2.1 ***Hybrid Label Index.*** Since we aim to find a path passing by a set of POIs, where the network distance between consecutive POIs is close to a user-specified distance, it is important to be able to compute the network distance between two POIs efficiently. As there is no network topology of POIs on the road network, we project POIs onto the nearest edges and consider the network distances between them. Next, we adopt 2-hop labeling [1, 2, 18] to compute network distances between vertices. Based on 2-hop labeling, we develop a POI label index to compute the network distances between POIs. The 2-hop label of a vertex is called a vertex label to distinguish it from a POI label.

Vertex label. We use 2-hop labeling because it enables state-of-the-art efficiency for computing distance in road networks. It constructs labels for vertices such that a distance query for any vertex pair (u, v) can be answered by only looking up the common vertices in their labels. For each vertex v, we precompute a label, denoted as $L(v)$, which is a set of label entries where each label entry is a pair $(o, d_G(v, o))$. We say that o is a pivot in a label entry if $(o, d_G(v, o)) \in L(v)$. Given two vertices u and v, we can find a common pivot o such that $(o, d_G(u, o)) \in L(u)$ and $(o, d_G(v, o)) \in L(v)$. Then we have:

$$d_G(u, v) = \min_{o \in V}\{d_G(u, o) + d_G(v, o)\} \tag{12}$$

Table 2: Vertex Labels

Vertex	Vertex label
v_1	$\{(v_1, 0), (v_3, 6), (v_5, 9)\}$
v_2	$\{(v_1, 7), (v_2, 0), (v_3, 5), (v_4, 9)\}$
v_3	$\{(v_3, 0)\}$
v_4	$\{(v_3, 8), (v_4, 0), (v_6, 7), (v_7, 11)\}$
v_5	$\{(v_3, 9), (v_5, 0), (v_7, 6)\}$
v_6	$\{(v_3, 5), (v_6, 0), (v_7, 4)\}$
v_7	$\{(v_3, 9), (v_7, 0)\}$
v_8	$\{(v_3, 11), (v_5, 4), (v_7, 2), (v_8, 0)\}$
v_9	$\{(v_3, 16), (v_5, 9), (v_7, 7), (v_8, 5), (v_9, 0)\}$
v_{10}	$\{(v_3, 12), (v_5, 16), (v_6, 7), (v_7, 11), (v_8, 12), (v_9, 7), (v_{10}, 0)\}$

Table 3: POI Labels

POI	POI label
p_1	$\{(v_1, 9), (v_2, 2), (v_3, 3), (v_4, 11)\}$
p_2	$\{(v_1, 2), (v_3, 8), (v_5, 7), (v_7, 13)\}$
p_3	$\{(v_3, 9), (v_4, 1), (v_6, 6), (v_7, 10)\}$
p_4	$\{(v_1, 7), (v_3, 11), (v_5, 2), (v_7, 8)\}$
p_5	$\{(v_3, 6), (v_6, 1), (v_7, 3)\}$
p_6	$\{(v_3, 10), (v_5, 1), (v_7, 5), (v_8, 3)\}$
p_7	$\{(v_3, 9), (v_5, 19), (v_6, 4), (v_7, 8), (v_8, 15), (v_9, 10), (v_{10}, 3)\}$

We say that the pair (u, v) is covered by o. The distance query $d_G(u, v)$ is then answered by o as the smallest sum of $d_G(u, o)$ and $d_G(v, o)$. Therefore, we can compute $d_G(u, v)$ in $O(|L(u)| + |L(v)|)$ time by using a merge-join like algorithm. Since all pairs of vertices are covered by the vertex labels, distance queries can be computed correctly.

POI label. In general, POIs occur infrequently in the road network. In order to reduce the space overhead, we only store labels that are related to POIs. Specifically, based on the vertex labeling, we propose a POI label index that enables network distance computation between POIs.

We denote the label of p_i by $L_p(p_i)$. Specifically, let e_i be the edge that p_i is located on, and let v_i, v_i' be the two endpoints of e_i. For each entry $(o, d_G(v_i, o)) \in L(v_i)$, we add an entry $(o, d_G(p_i.l, o))$ to $L_p(p_i)$, where $d_G(p_i.l, o) = d_G(p_i.l, v) + d_G(v, o)$. The label entries in $L(v_i')$ are processed in the same way. Note that if a vertex o is the pivot in both $L(v)$ and $L(v')$, i.e., $(o, d_G(v_i, o)) \in L(v_i)$ and $(o, d_G(v_i', o)) \in L(v_i')$, we only have to add an entry with the minimum distance, i.e., $(o, \min\{d_G(p_i.l, v_i) + d_G(v_i, o), d_G(p_i.l, v_i') + d_G(v_i', o)\})$, to $L_p(p_i)$. As a result, the network distance between two POIs p_i and p_j can be computed as follows.

$$d_G(p_i.l, p_j.l) = \min_{o \in V} \{d_G(p_i.l, o) + d_G(o, p_j.l)\},\ e_i \neq e_j, \quad (13)$$

where $(o, d_G(p_i.l, o)) \in L_p(p_i), (o, d_G(o, p_j.l)) \in L_p(p_j)$, and e_i, e_j are the road segments that p_i and p_j are located on, respectively.

Lemma 1. *POI labeling enables computing the network distance between any two POIs correctly.*

Proof. We know that the pivots in $L_p(p_i)$ are the union of the pivots in $L(v_i)$ and $L(v_i')$. The shortest path from p_i to p_j must pass through either endpoint of their road segments. Since the vertex labels of any two endpoints can compute the network distance between them correctly, the network distance between any two POIs can be computed correctly by the POI label. □

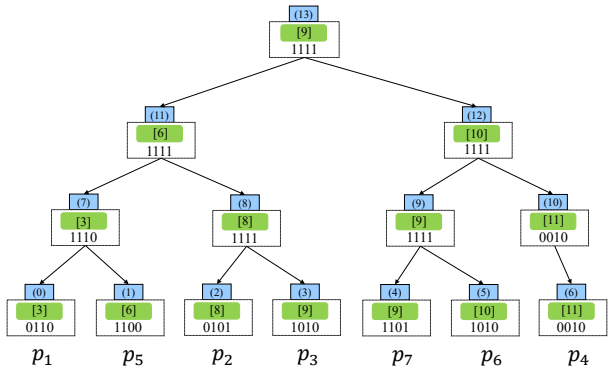


Figure 5: Structure of $PB(v_3)$

Example 3. *The vertex and POI labels for the example in Fig. 2 are shown in Tables 2 and 3, respectively. Suppose we are interested in $L_p(p_5)$. The endpoints of the edge that p_5 is located on are v_6 and v_7, and $d_G(p_5.l, v_6) = 1, d_G(p_5.l, v_7) = 3$. We first look up $L(v_6)$ and add $(v_3, 5+1), (v_6, 0+1), (v_7, 4+1)$ to $L_p(p_5)$. So far, $L_p(p_5) = \{(v_3, 6), (v_6, 1), (v_7, 5)\}$. Then we process $L(v_7)$. For the entry $(v_3, 9)$, we notice that v_3 is already in $L_p(p_5)$ and $9+3 > 6$, so we skip to the next entry $(v_7, 0)$. Since $0+3 < 5$, we update the distance between $p_5.l$ and v_7, and obtain $L_p(p_5) = \{(v_3, 6), (v_6, 1), (v_7, 3)\}$.*

4.2.2 ***Pivot Reverse POI Label.*** Suppose that we have already obtained a candidate POI p_{i-1} with respect to t_{i-1} in the template sequence and that we now aim to find the next candidate POI p_i that contains the keyword w_i and has the network distance $d_G(p_{i-1}.l, p_i.l)$ that is closest to the user specified distance d_i. Based on Eq. 13, we know that $d_G(p_{i-1}.l, p_i.l)$ can be divided into $d_G(p_{i-1}.l, o)$ and $d_G(o, p_i.l)$ by a common pivot o in $L_p(p_{i-1})$ and $L_p(p_i)$. The first term $d_G(p_{i-1}.l, o)$ can be obtained directly from $L_p(p_{i-1})$. However, we cannot obtain the second term since the target POI p_i is unknown. Therefore, we propose a pivot reverse (PR) POI label so that the second term and the target POI can be retrieved from $PR(o)$, where $PR(o)$ is the pivot reverse label of o. This label reverses the directions of all label entries $(o, d_G(p.l, o)) \in \cup L_p(p)$ that regard o as the pivot, i.e., $(p, d_G(p.l, o)) \in PR(o)$. For example, we have $(v_1, 9) \in L_p(p_1), (v_1, 2) \in L_p(p_2)$ and $(v_1, 7) \in L_p(p_4)$. For pivot v_1, we then have $PR(v_1) = \{(p_1, 9), (p_2, 2), (p_4, 7)\}$.

Based on the POI label and PR label indexes, we first find all pivots in $L_p(p_{i-1})$. Then, for each pivot o, we turn to $PR(o)$ to retrieve a matching POI p_i. To avoid enumerating all label entries of $PR(o)$, we organize the entries of $PR(o)$ in a B-tree (PB-tree), denoted as $PB(o)$.

PB-tree Construction. For each vertex $v \in V$, we construct a PB-tree $PB(v)$ that contains the information on keywords and network distances. The structure of $PB(v_3)$ is shown in Fig. 5.

Each PB-tree is a B^+-tree with fanout $f = 2$, and the structure is determined by the network distances in the PR index. To construct $PB(v)$, we first obtain a POI list S from $PR(v)$ that is sorted in ascending order of the network distance to v. Each leaf node in $PB(v)$ contains a POI $p \in S$ and the network distance $d_G(p.l, v)$.

We recursively generate non-leaf nodes in a bottom-up manner. The network distance of each non-leaf node equals the maximum network distance of its left subtree.

To store keyword information, we use a hash function H to map each keyword to a binary code with h bits. For a keyword w, one bit of $H(w)$ is set to 1. Therefore, the binary code of a leaf node that represents p is the superposition of $H(w)$ for all w that p contains, i.e., $H(p) = \vee_{w \in p.\Phi} H(w)$, where $p.\Phi$ is the keyword set of p. The binary code of each non-leaf node is the superimposition of those of its child nodes.

We observe that POIs are divided into fragments by utilizing the tree structure so that the network distances of the POIs in the same fragment are similar, which speeds up the retrieval of POIs by network distance. Meanwhile, the keyword information helps to filter out irrelevant fragments.

Basic Operations on the PB-tree. We discuss two operations on the PB-tree, called predecessor and successor operations.

(1) Given $PB(u)$ and a template $t(w, d)$, the predecessor operation (Pred()) finds a POI p that contains w and where $d_G(u, p.l)$ is no larger than and closest to d of all POIs in $PB(u)$. For a non-leaf node e, if $H(w) \wedge H(e) \neq 0$, we compare the network distance stored in e with d. If the network distance is no larger than d, we search its right subtree and then consider the left subtree if no suitable POI exists in the right subtree. If the network distance exceeds d, we only consider the left subtree. For a leaf node e, we directly check if e contains w and if $d_G(u, n.l)$ is no larger than d.

(2) Given $PB(u)$ and a template $t(w, d)$, the successor operation (Succ()) finds a POI p that contains w and where $d_G(u, p.l)$ is no smaller than and closest to d. For a non-leaf node e, if $H(w) \wedge H(e) \neq 0$, we continue to search its subtree. If the network distance stored in e is no smaller than d, we check if we can find a POI in its left subtree. If we can not, we turn to search the right subtree. Otherwise, we simply search its right subtree. A leaf node e that contains w and where $d_G(u, n.l)$ is no smaller than d is returned as result.

4.2.3 ***Branch and Bound Algorithm.*** Since the number of candidate paths increases exponentially with the number of templates, we develop a branch-and-bound (BAB) algorithm that avoids enumerating all candidate paths. The key idea of BAB is to utilize the upper bound of the route template distance so that the algorithm can terminate early.

Before we introduce the details of BAB, we present a greedy algorithm whose result can be used to initialize the upper bound UB. Given a query $Q = (l_0, \mathcal{T})$, we first add l_0 to the candidate path and then call a procedure called template-based POI search (TPS, details are presented in Section 4.2.4) that finds the next best matching POI p_1, where p_1 contains keyword w_1 and the distance d_R^1 between $(l_0, p_1.l)$ and t_1 is minimized. Then we insert $p_1.l$ into the candidate path and continue to find the next POI. We repeat this process until all mathcing POIs are found. Finally, we use the route template distance $d_R(\mathcal{P}, Q)$ of the path $\mathcal{P}$ returned by the greedy algorithm as the initial UB.

The algorithm's pruning condition works as follows. Assuming that we have obtained a partial candidate path $\mathcal{P}(l_0, p_1.l, ..., p_i.l)$, we call Procedure TPS to find a next best matching POI p_{i+1}. If the route template distance d_R^{i+1} between (p_i, p_{i+1}) and t_{i+1} exceeds UB, $p_i.l$ and $p_{i+1}.l$ can be removed safely from the current candidate path. Since p_{i+1} is the next best matching POI after p_i, it is impossible to obtain a complete candidate path that passes $p_i.l$ and has route template distance smaller than UB. When we find a complete candidate path, we update UB with its route template distance to the query.

Algorithm 1: Branch and Bound Algorithm

```
Input: Q = (l_0, T)
Output: An optimal CP with d_R(CP, Q)
Initialize stackP, stackD, tolerance parameter ε, and
  hierarchical lower bound θ;
stackP.push(l_0);
UB ← GreedyAlgorithm;
while stackP is not empty do
    i ← stackP.size();
    if TPS(p_{i-1}, d_i, w_i, θ, UB, ε) = true then
        Obtain p_i and d_R^i;
        θ ← 0.0;
        stackP.push(p_i.l);
        stackD.push(d_R^i);
        if i = k then
            if max{stackD} ≤ UB then
                UB ← max{stackD};
                CP ← stackP;
            Update θ by top of stackD;
            stackP.pop();
            stackD.pop();
    else
        Update θ by top of stackD;
        stackP.pop();
        stackD.pop();
return CP, d_R(CP, Q) ← UB;
```

For a query with $\mathcal{T} = \langle t_1(w_1, d_1), \ldots, t_k(w_k, d_k) \rangle$, BAB searches from the first to the k-th template to obtain a complete candidate path CP. First, for the user's location l_0, we generate its corresponding label $L_p(l_0)$ for computing the network distance, which is similar to the process of generating the POI label. We maintain two stacks, stackP and stackD, to store the current partial candidate path and the corresponding route template distance, respectively. Initially, we insert the origin l_0 into stackP. Then we continue to locate the remaining matching POIs by calling Procedure TPS.

When we obtain a partial candidate path $\mathcal{P}(l_0, p_1.l, \ldots, p_i.l)$, we call TPS to find the next matching POI p_{i+1}. Once p_{i+1} is found, we compute d_R^{i+1} between (p_i, p_{i+1}) and the template t_{i+1}. We consider two cases when verifying p_{i+1}:

(1) If d_R^{i+1} is no larger than the current UB, $p_{i+1}.l$ and d_R^{i+1} are inserted into stackP and stackD, respectively.

(2) If d_R^{i+1} exceeds the current UB, p_i satisfies the previously mentioned pruning condition, so we remove $p_i.l$ and d_R^i from stackP and stackD, respectively. We set a hierarchical lower bound θ to d_R^i. Unlike UB that is the upper bound between the query and the candidate path, θ is used to relax

the lower bound of the current d_R^i to find an alternative p_i' to p_i. The route template distance $d_R^{i'}$ between (p_{i-1}, p_i') and t_i is the minimum among all the untouched POIs. If p_i' is valid, we continue to find the subsequent POIs.

The pruning proceeds from low level to high level, that is, from k to 1, until the stacks become empty. The algorithm returns a complete candidate path and its corresponding route template distance.

4.2.4 ***Template based POI Search (TPS).*** TPS is applied to locate the next best matching POI p_{i+1} with respect to the template $t_{i+1}(w_{i+1}, d_{i+1})$. Based on whether the user specifies the distance d_{i+1}, we consider the following two cases to process the query.

(1) Specified distance. As mentioned in Section 4.2.3, we know that the route template distance d_R^{i+1} must be no larger than the current upper bound UB and no less than the hierarchical lower bound θ. Based on Eq. 1, we obtain the value range of $d_G(p_i.l, p_{i+1}.l)$:

$$\begin{gathered} lB \le d_G(p_i.l, p_{i+1}.l) \le lD \\ or \\ rD \le d_G(p_i.l, p_{i+1}.l) \le rB, \end{gathered} \tag{14}$$

where $lB = d_{i+1} - UB \cdot d_{i+1} \cdot \epsilon$, $lD = d_{i+1} - \theta \cdot d_{i+1} \cdot \epsilon$, $rD = d_{i+1} + \theta \cdot d_{i+1} \cdot \epsilon$ and $rB = d_{i+1} + UB \cdot d_{i+1} \cdot \epsilon$.

It is easy to see that the distance between POI pairs can be computed correctly through a common pivot o. In other words, $d_G(p_i.l, p_{i+1}.l)$ is divided into terms $d_G(p_i.l, o)$ and $d_G(o, p_{i+1}.l)$, where o is in $L_p(p_i)$ and $L_p(p_{i+1})$. Therefore, we first obtain the POI label $L_p(p_i)$ to get the pivot set. Since the POI label stores the distance from the POI to the pivot, for each vertex o in the pivot set, we compute the feasible range of $d_G(o, p_{i+1}.l)$ as follows.

$$\begin{gathered} lB_o \le d_G(o, p_{i+1}.l) \le lD_o \\ or \\ rD_o \le d_G(o, p_{i+1}.l) \le rB_o, \end{gathered} \tag{15}$$

where lB_o, lD_o, rD_o, and rB_o are obtained by subtracting $d_G(p_i.l, o)$ from lB, lD, rD, and rB. Note that, the smaller the route template distance is, the better the POI matches the query. Therefore, d_R^{i+1} between (p_i, p_{i+1}) and t_{i+1} is slightly larger than and closest to θ. As a result, it is better that $d_G(o, p_{i+1}.l)$ is closer to lD_o or rD_o within the range in Eq. 15. Thus, we utilize rD_o and w_{i+1} to apply a successor operation Succ() on $PB(o)$, while ensuring that $d_G(o, p_{i+1}.l)$ is no larger than rB_o. Next, a predecessor operation Pred() on $PB(o)$ is carried out with lD_o and w_{i+1} as input, while ensuring that $d_G(o, p_{i+1}.l)$ is no smaller than lB_o. We compare the route template distances that correspond to the results of Succ() and Pred(), and choose the smaller one to further reduce the search space by updating lB and rB. Then we continue to process the next pivot.

(2) Unspecified distance. When we explore the next best matching POI p_{i+1}, it is possible that the distance that corresponds to w_{i+1} is not extracted by RT-BERT and is thus missing. In this case, we aim to minimize the network distances for all templates without specified distances. The idea underlying unspecified distance search is similar to that underlying specified distance search. In the unspecified distance case, we still utilize the PB-tree to find the next POI. Procedure TPS finds the next candidate p_{i+1} that contains the query keyword w_{i+1} and where $d_G(p_i.l, p_{i+1}.l)$ is the smallest:

$$\min_{w_{i+1} \in p_{i+1}.\Phi} d_G(p_i.l, p_{i+1}.l) \tag{16}$$

Note that the sub-path returned by the algorithm contributes 0 to the overall route template distance of the complete candidate path.

Table 4: Route Description Sentence and Ground Truth.

Route description sentence	Ground truth
I want to find a route first passing a restaurant then walk about four hundred meters to an atm and another one kilometer to a fast food	O O O O O O O O O B-first.loc O O O B-second.dis I-second.dis I-second.dis O O B-second.loc O O B-third.dis I-third.dis O O B-third.loc I-third.loc
Get directions to the university five hundred meters away from me and then turn to a library six hundred meters away	O O O O B-first.loc B-first.dis I-first.dis I-first.dis O O O O O O O O B-second.loc B-second.dis I-second.dis I-second.dis O
Pick a route to the airport passing a fuel station	O O O O O B-second.loc O O B-first.loc I-first.loc
Search for a bar three kilometers away from here	O O O B-first.loc B-first.dis I-first.dis O O O

5 EXPERIMENTS

5.1 Experimental Settings

The RT-BERT model is implemented in Python3 with PyTorch, and the BAB algorithm is implemented in Java. All experiments are performed on a MacOS machine with an AMD 2.7GHz CPU and 8GB memory.

5.1.1 ***Datasets.*** To evaluate the RT-BERT model, we use two public datasets, ATIS [14] and Snips [6]. In addition, we generate a new dataset, called Route Description Natural Language (RDNL). The ATIS dataset is the most widely used single-domain dataset for natural language understanding tasks. It consists of audio recordings of people making flight reservations. ATIS contains around 5K sentences. The training, validation, and testing sets contain 4,478, 500, and 893 sentences, respectively, with a total of 21 intent types and 120 slot tags. The Snips dataset is collected using crowdsourcing through the SNIPS voice platform. Compared with the ATIS dataset, the Snips dataset is more complex due to its large vocabulary and cross-domain intents, making Snips a benchmark for natural language understanding evaluations. With Snips, we use 13,084 sentences for training, 700 for validation, and 700 for testing. The numbers of intent types and slot tags are 7 and 72, respectively.

To the best of our knowledge, no open dataset exists for route description natural language understanding. Therefore, we provide the dataset RDNL. This dataset is generated as follows.

(1) We write 300 route description sentences with different patterns and fill in templates with specific placeholders, such as 'w1' for the first POI keyword and 'd2' for the second specified distance. (e.g., Find a route where I go straight for about d1 passing by w1, then proceed about d2 to w2, and finally go d3 to reach w3).

(2) With these sentences as input, we fine-tune the GPT-2 [20] model to generate more expected route description sentences. To increase the diversity, the so-called temperature parameter that controls the randomness is set to 0.9.
(3) To ensure that the grammar is correct, we check all sentences generated by the GPT-2 model. A total of 3,976 sentences are obtained, and the intent labels of all sentences are annotated as 'SearchRoute'.
(4) We replace the placeholders in the generated sentences with words belonging to their respective categories. We first bind POI keyword placeholders, 'w1', 'w2', . . . , using check-in data[2]. For the placeholders 'd1', 'd2', . . . , we make random substitutions with distances in a specified range. (e.g., Find a route where I go straight for about fifteen kilometers passing by a restaurant, then proceed about eight kilometers to a university, and finally go five kilometers to reach an arts center).
(5) We annotate each sequence with a ground-truth sequence. In particular, we employ Begin-Inside-Outside (BIO) annotation that is also used in ATIS and Snips. More specifically, in the ground truth of a route description sentence, each word is labeled by 'O', 'B-X.Y', or 'I-X.Y' that represent an irrelevant noun phrase, the beginning of a template noun phrase, or the rest of a template noun phrase, respectively. X denotes the specified order of the template, and Y denotes the word as a POI keyword (loc) or a distance (dis) in the template. Thus, we replace the placeholders with corresponding slot tags in sentences and label all other words 'O'. Table 4 shows route description sentences and corresponding ground truth annotations, with keywords and distances in yellow and pink, respectively.
(6) To observe whether RT-BERT can recognize 'SearchRoute' sentences, we randomly add 'NotSearchRoute' sentences to the dataset. We annotate all words in these sentences as 'O', and their intent types are recorded as 'NotSearchRoute'. The RDNL dataset contains 4,900 sentences.
(7) We partition RDNL according to the ratios 8:1:1 to obtain training, validation, and testing sets that contain 3,824, 492, and 584 sentences, respectively. There are 15 slot tags and 2 intent types, i.e., 'SearchRoute' or 'NotSearchRoute'.

For the template-driven path query, we use two real datasets to evaluate the hybrid label index and the BAB algorithm. Shenzhen contains 1,943 POIs, 77,599 vertices, and 104,660 edges. New York contains 35,677 POIs, 451,631 vertices and 594,714 edges. All POIs are extracted from OpenStreetMap[3].

5.1.2 ***Baselines.*** We compare RT-BERT with the following.

- **Attention BiRNN:** Liu et al. [19] model intent classification and slot filling jointly using an attention based encoder-decoder neural network architecture.
- **Slot-Gated Full Attention:** Goo et al. [10] introduce a slot gate that focuses on learning the relationships between intent and slot attention vectors in order to obtain better semantic frame results by means of global optimization.
- **Joint BERT:** Chen et al. [5] propose a joint intent classification and slot filling model based on BERT that takes advantage of the correlation between intent classification and slot filling and of the strong generalization abilities of BERT.
- **SF-ID network:** E et al. [8] integrate the contexts of slots and intent using an attention mechanism and then present an SF-ID network to establish the direct connections between intents and slots.

Since there is no existing methods for TPQ, we use a dynamic programming algorithm as baseline which achieves an exact answer, but has low efficiency. The dynamic programming algorithm works as follows. For each keyword w, we create a list of POIs that correspond to w. Given a query $Q = (l_0, \mathcal{T})$, we retrieve the corresponding POI lists according to the order of the POI keywords in $\mathcal{T}$. Specifically, for POI p_i that contains w_i, we compute $D(w_i, p_i)$ that represents the minimum possible route template distance of a route that passes POIs with the keywords from w_1 to w_i consistent with the order in $\mathcal{T}$ and stops at $p_i.l$. After all keywords are processed recursively, we find the minimum $D(w_k, p_k)$ and backtrace through the corresponding POIs to construct a final path.

5.1.3 ***Parameter Settings.*** Our RT-BERT model is built on an English uncased BERT-Base model[4] that has 12 layers, 768 hidden states, and 12 heads. The maximum length of a sentence is 50, and the batch size is 128. During training, Adam [17] is applied for optimization with an initial learning rate of 5e-5.

For the template driven path query, Table 6 summarizes the parameter settings of the branch and bound algorithm. The default values are highlighted in bold. The ratio of UDT to SDT refers to the ratio of templates without specified distances to templates with specified distances.

5.2 Performance Evaluation

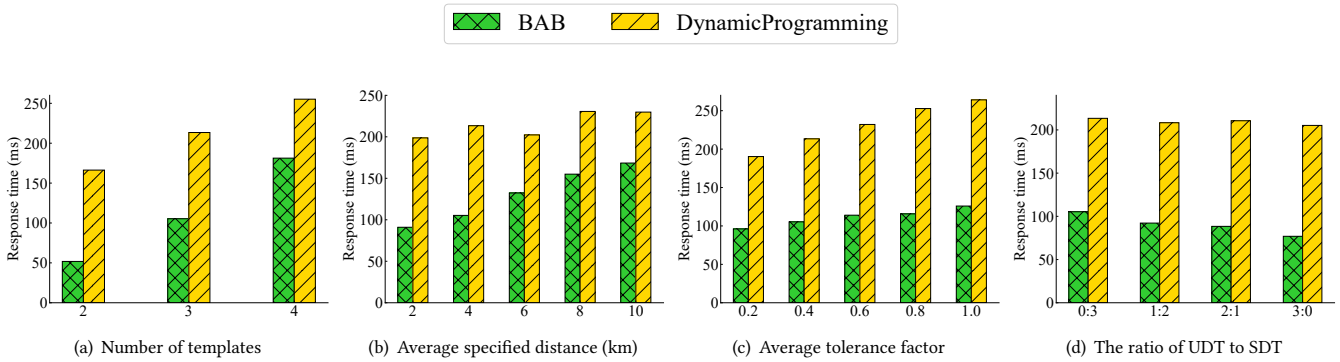
5.2.1 ***Route Description Language Understanding.*** To study the route description language understanding, we utilize three well-adopted metrics. The intent classification and slot filling tasks are evaluated using accuracy and the F1-score, respectively. In addition, the sentence-level semantic frame accuracy (sentence accuracy) is used to represent the overall performance of both tasks. It is defined as the proportion of the sentence whose slots and intents are both predicted correctly in the whole corpus. To evaluate the performance of RT-BERT, we first compare RT-BERT with baseline models. Then, we perform an ablation study to explore the contributions of key factors of the proposed model.

Overall results. Table 5 shows the experimental results. We observe that our RT-BERT model outperforms the previous state-of-the-art models on all three datasets, where all tasks (slot filling, intent classification, and semantic frame) show improvement. In particular, on the RDNL dataset, RT-BERT outperforms the baselines and achieves an intent classification accuracy of 99.90%, a slot filling F1-score of 94.81%, and a sentence-level semantic frame accuracy of 90.44%. We attribute the improvement of our model to the following reasons: 1) We use a pre-trained language model in RT-BERT that embodies general language knowledge. 2) The

[2]https://foursquare.com/
[3]https://www.openstreetmap.org/
[4]https://github.com/google-research/bert

Table 5: Comparison with Baselines

Method	ATIS			Snips			RDNL		
	Slot (F1)	Intent (Acc)	Sen. (Acc)	Slot (F1)	Intent (Acc)	Sen. (Acc)	Slot (F1)	Intent (Acc)	Sen. (Acc)
Attention BiRNN	94.20	91.10	78.90	87.80	96.70	74.10	91.52	98.82	84.67
Slot-Gated Full Attention	95.42	95.41	83.73	89.27	96.86	76.43	92.46	99.69	89.63
SF-ID Network	95.75	97.76	86.79	91.43	97.43	80.57	92.46	99.89	86.89
Joint BERT	96.10	97.50	84.20	97.00	98.60	92.80	93.76	99.90	84.91
RT-BERT	**97.18**	**98.02**	**89.84**	**97.61**	**98.84**	**93.92**	**94.81**	**99.90**	**90.44**



(a) Number of templates (b) Average specified distance (km) (c) Average tolerance factor (d) The ratio of UDT to SDT

Figure 6: Query Time

Table 6: Parameters of BAB

Parameters	Values
The number of templates	2, **3**, 4
Average distance (km)	2, **4**, 6, 8, 10
Average tolerance factor	0.2, **0.4**, 0.6, 0.8, 1.0
The ratio of UDT to SDT	**0:3**, 1:2, 2:1, 3:0

Table 7: Comparison with Variants of RT-BERT

Removed component	RDNL		
	Slot (F1)	Intent (Acc)	Sen. (Acc)
Self-attention	94.32	99.90	89.91
Intent-attention	94.41	99.90	90.02
BiLSTM	94.75	99.90	90.17
CRF	94.57	99.90	87.82
RT-BERT	**94.81**	**99.90**	**90.44**

intent-attention mechanism makes the intent classification and slot filling tasks highly correlated, and the intent classification boosts the slot filling. 3) We leverage a self-attention mechanism to capture global dependencies in a full sentence and learn the inner structural features of sentences in the slot filling subtask. 4) We utilize BiLSTM with CRF as our decoder, which makes efficient use of both past and future input features and also considers correlations between the labels in neighborhoods.

Ablation study. The experiments reported on so far demonstrate that the RT-BERT model is capable of significant improvements. To gain insight into the contribution of each part in the model for route description language understanding, we present a detailed ablation study using the RDNL dataset. Specifically, we compare the following variants of RT-BERT:

(1) Removing the self-attention
(2) Removing the intent-attention
(3) Replacing BiLSTM with a softmax layer
(4) Removing the CRF layer

The results are shown in Table 7. We have four observations. First, removing any component has no effect on the intent accuracy, but decreases the F1-score and sentence accuracy. That is because the components in our model are mainly designed for the slot filling subtask. Second, without self attention or slot attention, the performance of our model drops. We believe that the main reason is the lack of global dependency of the whole sentence and information interactions between intent and slot labels. Third, compared to using a softmax layer, use of the BiLSTM improves the performance of our model. We attribute this to the fact that BiLSTM is excellent in processing long-distance sequence information and can capture both forward and backward input features. Fourth, the CRF layer has a positive effect on the model performance. This is because the CRF layer can obtain the maximum possible sequence on sentence level.

5.2.2 ***Template Driven Path Query.*** To study the template-driven path query, we first compare the performance of the proposed algorithm with the dynamic programming algorithm in terms of time and space cost on the two datasets (New York and Shenzhen). Then we study the efficiency of both algorithms when varying important parameters. Due to the space limit, we only show the performance on the Shenzhen dataset in the second experiment, where the performance on New York is similar to that on Shenzhen.

Query set. We randomly generate 100 template driven path queries for each experiment. In the experiment of evaluating the overall performance of the algorithms, the parameters of template sequences are default values as mentioned in Section 5.1.3. To detect the influence of a certain parameter on the performance of the algorithms, we vary it and the other parameters are also set to their default values.

Overall performance. We first evaluate the query time, index size, and index construction time of the BAB and the dynamic programming algorithms. As vertex labeling performance has been studied in existing works, we only consider POI labeling and the PB-tree in our study. Table 8 shows that although the BAB algorithm has a larger index size and construction time due to using the PB-tree, it outperforms the dynamic programming algorithm clearly in terms of query time.

Varying the number of templates. Fig. 6(a) shows the query times of the algorithms when we vary the number of templates. Not surprisingly, the query time of both algorithms increase when the number of templates increases since more combinations are available to form candidate paths. With more templates, more iterations are triggered for the dynamic programming algorithm, and candidate POIs and paths increase for the BAB algorithm.

Varying the average specified distance. Fig. 6(b) shows the query times of the algorithms when we vary the average distance specified by users. We can see that the dynamic programming algorithm has little dependency on the query distance. The main reason is that it utilizes label to compute network distances between POIs, the query time of which only depends on the label size. For the BAB algorithm, one function of distance is to prune POIs when searching in the PB-tree. Therefore, if the distance is small, only few POIs are left after filtering which helps locate the next candidate more quickly.

Varying the tolerance factor ϵ. Fig. 6(c) reports the query times of the algorithms when varying the tolerance factor ϵ from 0.2 to 1.0. The time costs of both algorithms increase when we enlarge ϵ. The reason is that the tolerance factor determines the distance range when searching for the next matching POI. When ϵ becomes larger, the distance range is increased, and thus more POIs are considered as candidates.

Varying the ratio of UDT to SDT. We study the performance of the algorithms when processing unspecified distance templates by varying the ratio of UDTs to SDTs. As shown in Fig. 6(d), when we increase the ratio of UDTs to SDTs, the query time of the dynamic programming algorithm fluctuates slightly. That is because the algorithm needs to retrieve all POIs that contain the user-specified keyword for UDTs and SDTs. However, the BAB algorithm can not apply the pruning strategy when searching the matching POI without user-specific distance, which decreases the query time.

Performance of the greedy algorithm. For the greedy algorithm, we study the accuracy of its query results in addition the efficiency. The accuracy indicates how close it is to the optimal candidate path, which can be measured by two criteria: the matching ratio Δ_{match} and the hitting ratio Δ_{hit}:

$$\Delta_{match} = \frac{d_R(Q, \mathcal{P}_{greedy})}{d_R(Q, \mathcal{P}_{optimal})}, \tag{17}$$

Table 8: Performance Overview of Proposed Algorithms on QT (Query Time), IT (Index Time), and IS (Index Size)

Dataset		BAB	Dynamic programming
Shenzhen	QT (ms)	105.31	213.39
	IT (s)	1.88	0.23
	IS (MB)	37.86	11.02
New York	QT (ms)	367.02	877.86
	IT (s)	85.33	10.40
	IS (MB)	1369.14	294.21

Table 9: Performance of Greedy Algorithm on QT, IT, IS, MR (Matching Ratio), and HR (Hitting Ratio)

Dataset	QT (ms)	IT (s)	IS (MB)	MR	HR
Shenzhen	11.42	1.88	37.86	1.16	0.49
New York	39.07	85.33	1369.14	1.99	0.22

$$\Delta_{hit} = \frac{|\mathcal{P}_{greedy} \cap \mathcal{P}_{optimal}|}{|\mathcal{T}|}, \tag{18}$$

where $\mathcal{P}_{greedy}$ is the candidate path of greedy algorithm, $\mathcal{P}_{optimal}$ is the optimal candidate path, and $|\mathcal{T}|$ is the number of templates. Next, Δ_{match} represents the ratio of the route template distance of the greedy algorithm to the route template distance of the optimal route, while Δ_{hit} focuses on the percentage of matching POIs in $\mathcal{P}_{greedy}$ contained by $\mathcal{P}_{optimal}$. A smaller Δ_{match} and a larger Δ_{hit} yield better accuracy. The results in Table 9 show that the greedy algorithm has a short response time, but it offers no accuracy guarantees. On the New York dataset, both the efficiency and accuracy decrease due to its larger size and more candidate POIs.

6 CONCLUSION

We propose the SpeakNav framework that enables a new form of template driven path querying via a voice interface. Specifically, SpeakNav enables voice input of a sequence of keywords and associated distances. It then returns a route that, as best as possible, visits points of interest that match the keywords and are located according to the distances. To enable natural language interaction, SpeakNav provides a joint intent classification and slot filling model, called RT-BERT, that extracts keyword-distance pairs and distinguishes the order of them. Next, SpeakNav features a branch-and-bound algorithm and an accompanying PB-tree index that return accurate answers and also reduce the computational overhead. The experimental study shows that the proposed RT-BERT model outperforms the state-of-the-art approaches in terms of both intent classification and slot filling that relate to language understanding, and that the proposed branch-and-bound algorithm is capable of answering template driven path queries efficiently.

ACKNOWLEDGMENTS

This research is supported in part by the NSFC (Grants No. 61902134, 62011530437), the Hubei Natural Science Foundation (Grant No. 2020CFB871), and the Fundamental Research Funds for the Central Universities (HUST: Grants No. 2019kfyXKJC021, 2019kfyXJJS091).

REFERENCES

[1] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato Fonseca F. Werneck. 2011. A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks. In *SEA (Lecture Notes in Computer Science, Vol. 6630)*. Springer, 230–241.

[2] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD Conference*. ACM, 349–360.

[3] Lei Bi, Juan Cao, GuoHui Li, Nguyen Quoc Viet Hung, Christian S. Jensen, and Bolong Zheng. 2021. SpeakNav: A Voice-based Navigation System via Route Description Language Understanding. In *ICDE*. IEEE, 2669–2672.

[4] Xin Cao, Lisi Chen, Gao Cong, and Xiaokui Xiao. 2012. Keyword-aware Optimal Route Search. *PVLDB* 5, 11 (2012), 1136–1147.

[5] Qian Chen, Zhu Zhuo, and Wen Wang. 2019. BERT for Joint Intent Classification and Slot Filling. *CoRR* abs/1902.10909 (2019).

[6] Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, Maël Primet, and Joseph Dureau. 2018. Snips Voice Platform: an embedded Spoken Language Understanding system for private-by-design voice interfaces. *CoRR* abs/1805.10190 (2018).

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*. 4171–4186.

[8] Haihong E, Peiqing Niu, Zhongfu Chen, and Meina Song. 2019. A Novel Bi-directional Interrelated Model for Joint Intent Detection and Slot Filling. In *ACL (1)*. Association for Computational Linguistics, 5467–5471.

[9] Hiroshi Fukui, Tsubasa Hirakawa, Takayoshi Yamashita, and Hironobu Fujiyoshi. 2019. Attention Branch Network: Learning of Attention Mechanism for Visual Explanation. In *CVPR*. Computer Vision Foundation / IEEE, 10705–10714.

[10] Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and Yun-Nung Chen. 2018. Slot-Gated Modeling for Joint Slot Filling and Intent Prediction. In *NAACL-HLT (2)*. Association for Computational Linguistics, 753–757.

[11] Daniel Guo, Gökhan Tür, Wen-tau Yih, and Geoffrey Zweig. 2014. Joint semantic utterance classification and slot filling with recursive neural networks. In *SLT*. IEEE, 554–559.

[12] Patrick Haffner, Gökhan Tür, and Jerry H. Wright. 2003. Optimizing SVMs for complex call classification. In *ICASSP (1)*. IEEE, 632–635.

[13] Dilek Hakkani-Tür, Gökhan Tür, Asli Çelikyilmaz, Yun-Nung Chen, Jianfeng Gao, Li Deng, and Ye-Yi Wang. 2016. Multi-Domain Joint Semantic Frame Parsing Using Bi-Directional RNN-LSTM. In *INTERSPEECH*. ISCA, 715–719.

[14] Charles T. Hemphill, John J. Godfrey, and George R. Doddington. 1990. The ATIS Spoken Language Systems Pilot Corpus. In *HLT*. Morgan Kaufmann.

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

[16] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *EMNLP*. ACL, 1746–1751.

[17] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR (Poster)*.

[18] Ye Li, Leong Hou U, Man Lung Yiu, and Ngai Meng Kou. 2017. An Experimental Study on Hub Labeling based Shortest Path Algorithms. *Proc. VLDB Endow.* 11, 4 (2017), 445–457.

[19] Bing Liu and Ian Lane. 2016. Attention-Based Recurrent Neural Network Models for Joint Intent Detection and Slot Filling. In *INTERSPEECH*. ISCA, 685–689.

[20] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[21] Suman V. Ravuri and Andreas Stolcke. 2015. Recurrent neural network and LSTM models for lexical utterance classification. In *INTERSPEECH*. ISCA, 135–139.

[22] Christian Raymond and Giuseppe Riccardi. 2007. Generative and discriminative algorithms for spoken language understanding. In *INTERSPEECH*. ISCA, 1605–1608.

[23] Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.* 45, 11 (1997), 2673–2681.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NIPS*. 5998–6008.

[25] Andrew J. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inf. Theory* 13, 2 (1967), 260–269.

[26] Yude Wang, Jie Zhang, Meina Kan, Shiguang Shan, and Xilin Chen. 2020. Self-Supervised Equivariant Attention Mechanism for Weakly Supervised Semantic Segmentation. In *CVPR*. IEEE, 12272–12281.

[27] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR* abs/1609.08144 (2016).

[28] Bin Yao, Mingwang Tang, and Feifei Li. 2011. Multi-approximate-keyword routing in GIS data. In *GIS*. ACM, 201–210.

[29] Kaisheng Yao, Geoffrey Zweig, Mei-Yuh Hwang, Yangyang Shi, and Dong Yu. 2013. Recurrent neural networks for language understanding. In *INTERSPEECH*. ISCA, 2524–2528.

[30] Sen Zhao, Lei Zhao, Sen Su, Xiang Cheng, and Li Xiong. 2018. Group-based keyword-aware route querying in road networks. *Inf. Sci.* 450 (2018), 343–360.

[31] Bolong Zheng, Han Su, Wen Hua, Kai Zheng, Xiaofang Zhou, and Guohui Li. 2017. Efficient Clue-Based Route Search on Road Networks. *TKDE* 29, 9 (2017), 1846–1859.