HypGen: A Generative Approach for Natural Language Understanding Enrichment Using Implicit User Feedback

Anonymous ConvAI 2022 submission

Abstract

In a conversational AI system, Natural Language Understanding (NLU) is a key component that produces the semantic interpretations for a user request. As one of the most upstream components, NLU errors are costly and have a wide blast radius to the end-to-end system behavior. We propose a model-based generative approach for automatic NLU n-best enrichment, by leveraging implicit user feedback learned from the existing user dissatisfaction and rephrase detection techniques. It aims to generate reasonable and promising NLU interpretation candidates that can potentially correct NLU errors. We propose three different modeling designs and compare their performance with extensive experiments on live traffic from one of today's state-of-the-art large-scale conversational AI systems.

1 Introduction

Natural Language Understanding (NLU) is a core component in a conversational AI system (Kepuska and Bohouta, 2018), which produces semantic interpretations of a user request. Typically, NLU involves a set of sub-tasks including domain/intent classification and slot filling (El-Kahky et al., 2014). For instance, given a request *"Play a Christmas song"*, NLU can produce an interpretation related to *Music* domain, with an intent to *Play Song* and the *Genre* slot filled with *Christmas* for entity resolution. As one of the most upstream components in the conversational AI workflow (Sarikaya, 2017), NLU's outputs are used by a series of subsequent downstream components, such as dialog management, routing logic to back-end domain applications, and natural language generation.

Each core component within a conversational AI system, including NLU, usually gives its outputs in the form of n-best predictions (Hakkani-Tür et al., 2006; Williams and Balakrishnan, 2009). The top prediction is not always correct and thus outputting the n-best predictions increases the chances of propagating the correct prediction to downstream. In case the top prediction is unable to deliver satisfactory end-to-end user experience, the n-best can be used by downstream (e.g., domain application) for recovering from errors or selecting the most appropriate prediction in a late-binding fashion if more context is needed to better disambiguate. The n-best can also be used for exploration (Qi et al., 2018) and downstream tasks' quality improvements (Li et al., 2020).

In practice, NLU often fails to contain the correct interpretation in the n-best for a variety of reasons. First, the training data for NLU largely comes from manual annotations, which can be error-prone. Also, the challenge of multi-label annotation makes it more difficult to comprehensively cover the user context and possible interpretations. Another limitation comes from the discrepancy in the data distributions compared to real traffic, mainly caused by speech recognition errors and constantly changing user behavior. One way to partially address these challenges is to continuously sample live traffic, manually annotate NLU errors to generate new supervision data and update the NLU models, which is still a costly manual effort.

We propose a model-based generative solution, Hypothesis Generator (HypGen), for automatic NLU n-best enrichment by leveraging implicit user feedback. By enrichment, we mean generating and then injecting (or identifying if already present) a promising or relevant interpretation in the n-best that can potentially correct the corresponding NLU error or deliver a better end-to-end user experience. We base on a scalable approach using two implicit user feedback signals, dissatisfaction and rephrasing, where the former is for detecting user dissatisfaction and the latter for user's correcting behavior (Park et al., 2021). For instance, as shown in Figure 1, a dissatisfied user can explicitly ask the system to stop its current response or action,

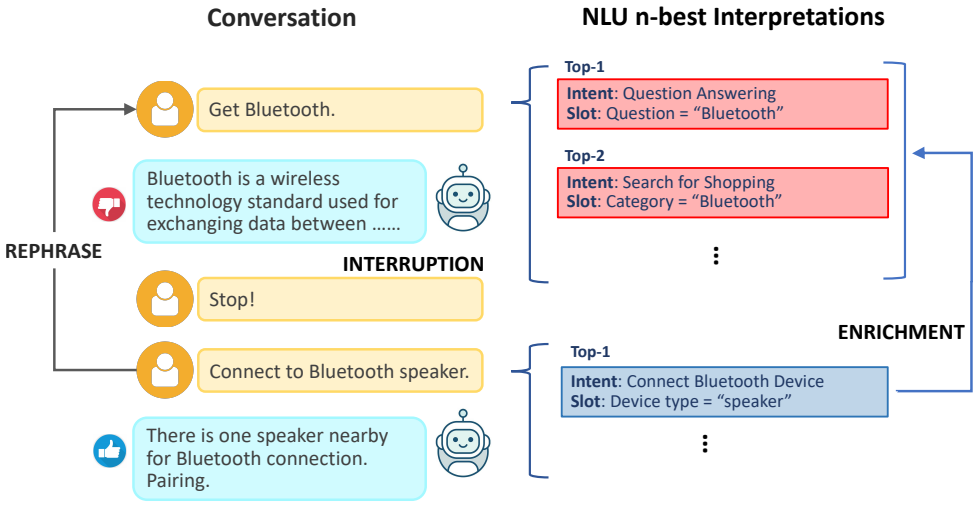


Figure 1: An example scenario, on how implicit user feedback can be used to automatically detect user dissatisfaction, which can be corrected by learning from the user rephrase behavior to generate and enrich the original NLU outputs with a more relevant interpretation.

and they can rephrase the request with less room for ambiguity. We can use the rephrased request's NLU interpretations to learn how to enrich the interpretations for the original request.

The contribution of this work is threefold. First, to our knowledge, this work is the first in the literature introducing the problem of NLU n-best enrichment in the context of large-scale personal assistant and conversational AI systems in production. Second, we propose an automatic and generative approach for enriching NLU n-best by leveraging implicit user feedback, without manual annotations. Last, we show the performance of our approach with extensive experiments on live production traffic from one of today's state-of-the-art large-scale conversational AI systems.

2 Background and related work

Implicit user feedback has been shown to be useful for many tasks in recommendation systems (Rendle et al., 2012; He and McAuley, 2016) and search engines (Joachims, 2002; Bi et al., 2019). For example, browsing, viewing, clicking or purchase history can be employed to enhance the quality of recommendation. There have been a small, similar set of pioneering work in the context of conversational AI systems on leveraging implicit feedback. An early attempt is to detect a user sentiment and adjust the AI response in open conversation setting (Zhang et al., 2018), which did not necessarily intend to clarify user intent. In terms of clarifying or correcting NLU interpretation, request text rewrite (Ponnusamy et al., 2019) and music entity relabeling (Muralidharan et al., 2019) are a few representative related works. One of the most recent effort was to alter the NLU interpretation's downstream ranking model behavior using implicit feedback (Park et al., 2021). However still, there has been no significant effort directly improving or enriching the NLU outputs in the current literature.

This work leverages two specific types of implicit user feedback signals, *user dissatisfaction* and *user rephrases*. This is based on our insight that it is often times possible to accurately infer user dissatisfaction by analyzing the interaction between the conversational system and the user. For instance, the system response can be generic error handling such as "Sorry, I did not understand," or the user can explicitly interrupt the system by expressing dissatisfaction such as "Stop" (although not always the case, the users are dissatisfied in most of the cases according to our user study). In such cases, users often clarify their request by rephrasing it in clearer terms, and the information can be used to infer their true intention associated with the original request that ended up with dissatisfaction. The following subsections briefly introduce the related enabling techniques to automatically (1) infer user dissatisfaction given a request and (2) infer whether there is a follow-up rephrase.

2.1 User Dissatisfaction Detection

Most user feedback is implicit, unless otherwise specifically solicited. There are many promising implicit user behavior signals that can be employed for the detection of user dissatisfaction during inter-

action with a conversational AI system. To name a few, termination (cancelling a conversation or experience), abandonment (leaving without completing a conversation), interruption (barging in while the system is still giving a response), error-correcting language (following up with "no, ..." or "I mean, ..."), negative sentiment language showing frustration (Beaver and Mueen, 2020; Sarikaya, 2017) are useful signals for user dissatisfaction detection.

Apart from the implicit user behavior signals, there are other useful signals from the system response. They include generic error-handling system responses ("I don't know that one."), natural language error-handling responses (when a media entity is not found), and an absence of a response (Beaver and Mueen, 2020; Sarikaya, 2017). Component-level signals such as latency or confidence scores for the underlying models, e.g., ASR or NLU, are useful as well. We may also apply advanced modeling strategies taking into account user interaction patterns and use past interaction history for additional context (Bodigutla et al., 2020), or contextual metrics relating to usage scenario, e.g., use dwell time in search for listening-to-music context (Kiseleva et al., 2016).

2.2 User Rephrase Detection

The major body of works in this literature has been approaching from the perspectives of sentence semantic similarity and paraphrase detection. The representative achievements are made through latent semantic analysis (Landauer et al., 1998), lexical matching (Manning and Schutze, 1999), leveraging meaning or concepts of words with knowledge bases such as WordNet (Mihalcea et al., 2006), and word (Camacho-Collados and Pilehvar, 2018) and sentence embeddings (Reimers et al., 2019). Regarding model architecture, Siamese network has been frequently applied with CNN (Hu et al., 2014), LSTM (Mueller and Thyagarajan, 2016), and BERT (Reimers et al., 2019). It is also related to the community question-answering systems in finding semantically similar topics (Srba and Bielikova, 2016).

3 Problem Definition

The goal is to generate a new, reasonable NLU interpretation candidate from an existing user request and corresponding NLU interpretations, identified through the aforementioned implicit user feedback detection techniques. Formally, denote a tuple $f_i = (t_i, C_i, H_i)$ representing the features for a user request r_i, where t_i is the request text of r_i, C_i is a set of contextual features (e.g., device has a screen), and $H_i = [h_{i,1}, h_{i,2}, \dots, h_{i,n}]$ is a list of n-best hypotheses produced from the existing NLU. Here, each $h_{i,j} = (q_{i,j}, S_{i,j})$ is a pair of intent $q_{i,j}$ and a list of slots $S_{i,j} = [s^1_{i,j}, s^2_{i,j}, \dots]$ with each slot $s^l_{i,j} = (k^l_{i,j}, v^l_{i,j})$ a pair of slot key and value, respectively. The goal is then to take f_i as an input and to generate the ground-truth hypothesis $g_i = (q^g_i, S^g_i)$ as an output.

4 Solution Architecture: HypGen

We propose HypGen, consisting of two modules : a user request feature encoder m_{enc} and a new hypothesis generation decoder m_{dec}, as shown in Figure 2. Note that the decoder can have a few different designs in a way of incorporating the relationships between hypothesis elements: domain, intent, and slots, and the figure describes one possible design among them. Formally, given the features f_i for a request, the encoder m_{enc} takes f_i as input to generate a single vector $z_i = m_{enc}(f_i)$ that summarizes the information in f_i. Then, the decoder m_{dec} takes z_i as input and aims to generate the ground-truth hypothesis $g_i = (q^g_i, S^g_i)$.

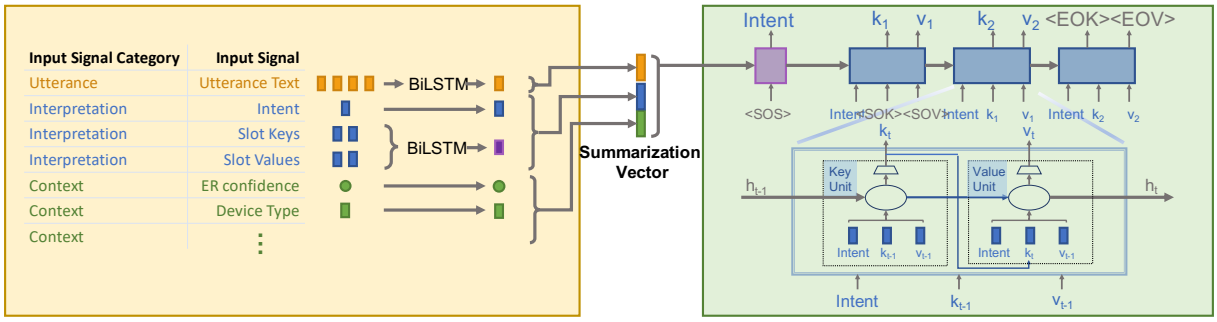
4.1 Encoder

The encoder summarizes the three input signal subcomponents into one vector in the following ways.

Request Text Encoding: Given the text t_i, we first tokenize it to get a list of word tokens $W_i = [w_1, w_2, \dots, w_m]$. Then, for each word w_i, we convert it into the corresponding embedding vector to get a list of embeddings $E_i = [e^t_1, e^t_2, \dots, e^t_m]$. A Bi-LSTM (Hochreiter and Schmidhuber, 1997) is applied to E_i to generate the text encoding z^t_i.

Context Encoding: The request context C_i can contain features that are either categorical (e.g., device type) or numerical (e.g., model confidence score). For each context feature $c_j \in C_i$, if c_j is categorical, we represent it with an embedding vector e^C_j. Otherwise, we set $e^C_j = c_j$ by directly using the numeric value. We generate the context encoding $z^C_i = \bigoplus_j e^C_j$ by concatenating the representations for all context features.

NLU Hypothesis Encoding: Given the NLU n-best hypotheses $H_i = [h_{i,1}, h_{i,2}, \dots, h_{i,n}]$, there are different ways to encode H_i. In this work, we encode H_i by only summarizing the information



(a) Encoder (b) Decoder

Figure 2: The overall HypGen architecture following the encoder-decoder framework - (a) The encoder architecture is shown on the left and (b) one possible decoder architecture, HIERARCHICAL design, is shown on the right.

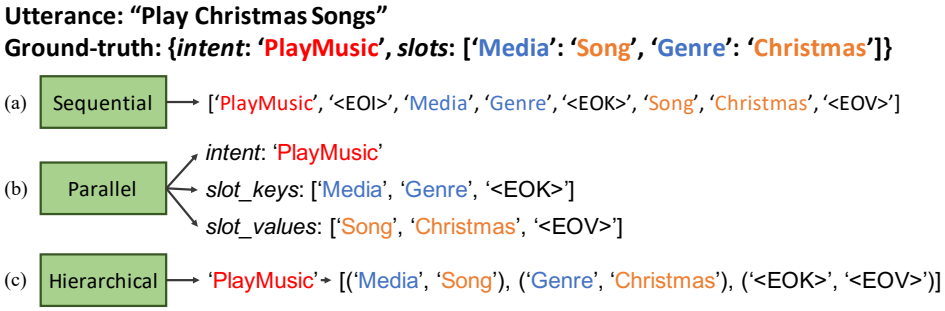


Figure 3: Examples for three different HypGen decoding designs: SEQUENTIAL, PARALLEL, and HIERARCHICAL.

in the 1-best hypothesis $h_{i,1}$, which is the most critical, and we leave it as future work on leveraging the full n-best information for more effective encoding. To encode $h_{i,1} = (q_{i,1}, S_{i,1})$, we first convert the intent $q_{i,1}$ into an embedding e_i^q. Then, we use a Bi-LSTM to summarize the slot key list $[k_{i,1}^1, k_{i,1}^2, \dots]$ of $S_{i,1}$ and another to summarize the slot value list $[v_{i,1}^1, v_{i,1}^2, \dots]$ to get the embeddings e_i^k and e_i^v. Finally, we concatenate all embeddings to get the hypothesis encoding $z_i^H = e_i^q \oplus e_i^k \oplus e_i^v$.

Final summarization: For the encodings z_i^t, z_i^C, and z_i^H for all three feature parts, the final summarization encoding $z_i = z_i^t \oplus z_i^C \oplus z_i^H$ is obtained by their concatenation.

4.2 Decoder

An NLU hypothesis is a complex yet structured result, with each element (i.e., intent, slot key, slot value) having a clear semantic meaning. Furthermore, these elements have inherent relationship. Certain slot keys may only appear given a specific intent, and similarly, certain slot values correspond to specific slot keys. Leveraging this insight, we describe three different modeling designs for the HypGen decoder, namely SEQUENTIAL, PARALLEL, and HIERARCHICAL.

4.2.1 Sequential Decoder

Motivated by Seq2seq (Sutskever et al., 2014) decoding, a sequential decoder, called SEQUENTIAL, generates hypotheses with each hypothesis simply a sequence of tokens (see Figure 3(a)). Specifically, given the summarization vector z_i from the encoder, an LSTM uses z_i as the initial hidden state and decodes a list of tokens serially where each token can be any element type (intent, slot key, or slot value) comprising a hypothesis. Since a hypothesis is structured, we explicitly have SEQUENTIAL first generate an intent, followed by a list of slot keys and then a list of slot values. To be able to separate the hypothesis element types, we use special tokens as sentinels for each element type generation, which are <*EOI*> (i.e., end of intent), <*EOK*> (i.e., end of slot keys), and <*EOV*> (i.e., end of slot values).

In the model training stage, we pre-process the ground-truth hypothesis of each training instance to a list of tokens conforming to the aforementioned format. For inference, we use a standard k-beam search to generate the k-best candidates. SEQUENTIAL is a simple and straight-forward baseline de-

sign for HypGen decoding.

4.2.2 Parallel Decoder

A parallel decoder, called PARALLEL, generates hypotheses with a separate decoder for generating each element type of intent, slot key, and slot value (see Figure 3(b)). Specifically, given the summarization vector z_i, the intent decoder takes z_i as input and performs a multi-class classification with a 2-layer feed-forward network to generate the intent. In parallel, two distinct LSTMs also take z_i as the initial hidden state vector and decode the slot keys and values respectively.

In the model training stage, we pre-process the ground-truth hypothesis of each training instance by splitting it into each element type. For inference, we adopt a modified k-beam search as follows. Given the summarization vector z_i, we first apply the intent decoder to get the top-k intents T_i with the highest confidence scores. Then, we perform a standard k-beam search on the slot key decoder and also on the slot value decoder separately to generate the top-k slot key sequence L_i and value sequence V_i with the highest confidence scores. Finally, we generate the k-best hypothesis candidates G_k from all possible combinations of $T_i \times L_i \times V_i$ with the highest confidence scores. Compared to SEQUENTIAL, PARALLEL is architecturally more complex, by modeling the structure of a hypothesis with three explicit subsections.

4.2.3 Hierarchical Decoder

As shown in Figure 2(b), a hierarchical decoder, called HIERARCHICAL, generates hypotheses with two decoders: an intent decoder and a slot decoder with a key-value (KV) cell (see Figure 3(c)). Given the summarization vector z_i, an LSTM-based intent decoder takes z_i as the initial hidden state vector. It performs a one-step decoding to generate the intent q_i and the hidden state vector h_i^q. The slot decoder, which uses h_i^q as the initial hidden state vector, produces a sequence of slot key-value pairs.

In detail, the slot decoder generates a sequence of slot key-value pairs recurrently with the KV-cell. As shown in Figure 2(b), the KV-cell extends LSTM and has two connected units: a key unit and a value unit. For the t-th step, the **key unit** takes the generated intent q and the previously generated key-value pair (k_{t-1}, v_{t-1}) as input to predict the slot key k_t. Specifically, denote x_t^k to be the embedding concatenation of q, k_{t-1}, and v_{t-1}. Also denote h_{t-1} to be the previous hidden state. Then we pass x_t^k and h_{t-1} to an LSTM to predict the output o_t^k and hidden state h_t^k, where o_t^k will be used to generate the slot key k_t as follows:

$$\begin{aligned} o_t^k, h_t^k &= \text{LSTM}^k(x_t^k, h_{t-1}), \\ P(k_t) &= \text{softmax}(W^k o_t^k + b^k), \end{aligned} \tag{1}$$

Once we have k_t, the **value unit** takes the hidden state h_t^k, the embedding concatenation x_t^v of the intent q, the newly generated key k_t and the previous value v_{t-1} as input, and use another LSTM to predict the output o_t^v and hidden state h_t, where o_t^v is used to generate the slot value v_t as follows:

$$\begin{aligned} o_t^v, h_t &= \text{LSTM}^v(x_t^v, h_t^k), \\ P(v_t) &= \text{softmax}(W^v o_t^v + b^v), \end{aligned} \tag{2}$$

Similarly to PARALLEL, a modified k-beam search is used for inference. Specifically, given the vector z_i, we apply the intent decoder to get the top-k intent predictions T_i. For each $t_j \in T_i$, we run a standard k-beam search on the slot decoder with input t_j and select the 1-best slot sequence S_j with the highest confidence. Finally, we generate the k-best hypothesis candidates $G_k = \{(t_j, S_j) \mid t_j \in T_i\}$. Compared with the other two models, HIERARCHICAL more clearly captures the hypothesis structure among the element types (e.g., slot key-value bond) and thus has the most complex architecture.

4.3 Supervision Data

We leverage the user dissatisfaction and user rephrase signals to prepare the training data for HypGen. Denote $\mathcal{S}_\Delta = \{s_1, s_2, \ldots, s_n\}$ to be the user interaction session data from the production log for a certain period of time Δ. Each session $s_i = \{(r_1, a_1), (r_2, a_2), \ldots, (r_q, a_q)\}$ is a list of time-consecutive request-action pairs (i.e., r_j is a user request, a_j is the corresponding conversation system action or response) that are temporally close to each other. Given a session $s_i \in \mathcal{S}_\Delta$, we use the analysis tool f_{impl}[1] that (1) detects if a_j caused end-to-end user dissatisfaction for r_j for each $(r_j, a_j) \in s_i$, and (2) whether there exists $(r_k, a_k) \in s_i$ with $k > j$ such that r_k is a semantic rephrase of r_j with a_k not have caused user dissatisfaction. Based on this, we generate the training data $\mathcal{D}_\Delta$ as follows.

First, we apply f_{impl} to each session $s_i \in \mathcal{S}_\Delta$ to extract all rephrase pairs R_i^s in s_i with each

[1] In today's production system, f_{impl} show F_1 scores over 0.70 for detecting user dissatisfaction as well as user rephrase.

Datasets	Size	Data Properties	Size
Train_1	> 10MM	# of intents	> 100
Test_1	> 800K	# of slot keys	> 100
Test_2	> 350K	# of slot values	> 100K

Table 1: Data statistics. Train_1 and Test_1 are a typical dataset split into training and testing from live traffic log.Test_2 is a subset of Test_1, where the original n-best does not contain the correct interpretation.

Metric		Model		
		Sequence	Parallel	Hierarchical
Intent	Accuracy	85.84	85.19	85.77
Slot keys	Precision	94.85	57.75	94.52
	Recall	91.21	66.68	93.33
	F_1	92.83	61.38	93.82
Slot values	Precision	91.16	48.73	83.33
	Recall	42.69	53.62	81.99
	F_1	56.65	50.42	82.55
Hypothesis	Accuracy	58.8	34.53	75.27
	Hit@k	58.84	34.6	93.75

Table 2: Hypothesis generation result on Test_1.

pair $(r_j, r_k) \in R_i^s$ following (1) r_k is a semantic rephrase of r_j with $k > j$ and (2) a_k not showing user dissatisfaction while a_j showing dissatisfaction. Next, for each $(r_j, r_k) \in R_i^s$, we extract the features $f_j = (t_j, C_j, H_j)$ for r_j. Denote g_k to be the NLU hypothesis for r_k that is selected and executed by the conversational system end-to-end. We infer that g_k could be a correct hypothesis for r_j since g_k's end results do not show user dissatisfaction and r_k rephrases r_j. Based on session s_i, we generate a set of training instances $D_i^s = \{(f_j, g_k) \mid (r_j, r_k) \in R_i^s\}$. We note that it is possible for the rephrase interpretation g_k to be the same as the executed NLU hypothesis for r_i. In this work, we keep all such instances in our data, to not bias our model to learn to generate hypotheses that are different from r_i's original NLU selection when the original NLU hypothesis is correct. Finally, we union the data points from all sessions to construct the complete supervision data $\mathcal{D}_\Delta = \bigcup_i D_i^s$.

5 Experiments

Datasets and Experiment Settings: We took live traffic from a commercial large-scale conversational AI system in production, processed the data so that users are not identifiable, and generated our supervision data. We split the data into the training set Train_1 and test set Test_1. We also extracted a subset Test_2 from Test_1, in which each instance in Test_2 has the ground-truth (i.e., correct) interpretation that is not in the request's n-best interpretation hypotheses, which is a more challenging scenario. See Table 1 for the statistics.

Metric		Model		
		Sequence	Parallel	Hierarchical
Intent	Accuracy	+0.0	-0.4	**+0.1**
Slot keys	Precision	+0.0	-38.5	-0.4
	Recall	+0.0	-24.5	**+2.7**
	F_1	+0.0	-32.3	**+1.2**
Slot values	Precision	+0.0	-47.9	**+1.2**
	Recall	+0.0	+130.4	**+292.8**
	F_1	+0.0	+39.2	**+156.2**
Hypothesis	Accuracy	+0.0	-54.9	**+105.6**
	Hit@k	+0.0	-53.3	**+145.3**

Table 3: Hypothesis generation performance result on Test_2. It is more challenging task as the input n-best does not have the correct interpretation.

We used NLTK (Bird et al., 2009) to tokenize the request and used pre-trained GloVe (Pennington et al., 2014) for encoder word embeddings. All models were implemented in PyTorch, trained and evaluated on AWS p3.8xlarge instances with Intel E5-2686 CPUs, 244 GB memory, and 4 Nvidia V100 GPUs. We used Adam (Kingma and Ba, 2015) with a learning rate of 1e-3 as the optimization algorithm. The models were trained for 4 epochs with a 16 batch size. We used a teacher-forcing rate of 0.5 in training and a beam size of 5 for model inference.

5.1 Overall Evaluation

This section describes the overall generation performance of the three proposed model variants. Our evaluation consists of two parts for each variant: component-level and hypothesis-level. The component-level evaluation shows the generation performance for each component in a hypothesis. To this end, we report the *accuracy* of the generated intent, as well as *precision*, *recall* and F_1 score for both slot key and slot value generation. The hypothesis-level evaluation shows the full hypothesis generation performance. We use two metrics: *accuracy of the top-1 generation* and *hit@k* after running a k-beam search (i.e., whether there is a hypothesis in the top-k generations identical to the ground-truth), where k = 5. For all evaluations, we use the performance of SEQUENTIAL

as the baseline and show the relative changes for all metric scores for PARALLEL and HIERARCHICAL[2]. For reference, we show the absolute score ranges for SEQUENTIAL on key metrics. On $Test_1$, component-wise SEQUENTIAL achieves >80% on both intent accuracy and slot key generation F_1, and >50% on slot value generation F_1. Hypothesis-wise, it achieves >50% for both hypothesis generation accuracy and hit@k. For $Test_2$, it has >70% for intent accuracy and slot key F_1, and >20% for slot value F_1, while the hypothesis generation accuracy and hit@k are >20%. Note that in production systems, if we have at least the intent generated correctly, it is straight-forward to extract the top full NLU hypothesis (intent, slot keys, and slot values) corresponding to the intent, which can significantly complement relatively lower performance on slots.

Table 2 compares the overall performance of different modeling variants on $Test_1$. First, for intent generation, all model variants achieve comparably high accuracy. This is because SEQUENTIAL and HIERARCHICAL predict the intent as the first token in the output while PARALLEL has a separate classifier for intents, making it less challenging. Second, when generating slot keys, the three models' scores diverge. Specifically, HIERARCHICAL achieves the best performance, while PARALLEL shows the worst (i.e., 33.8% lower than SEQUENTIAL regarding F_1). This suggests that it is critical to model the relationship between intent and slot keys – HIERARCHICAL and SEQUENTIAL implicitly do so by generating slot keys conditioned on the intent value while PARALLEL uses a separate decoder for slot keys, unconditioned on the intent value. Third, slot value generation task is more challenging compared to slot key, as there are much more variety of possible slot values than keys (see Table 1), making it more challenging to generate. However still, HIERARCHICAL outperforms the rest by a large margin. This is because the key-value cell tightly bonds a slot value to its key, narrowing down the scope of the problem.Finally, for hypothesis-level evaluation, we see that HIERARCHICAL outperforms both SEQUENTIAL (i.e., by 28.0% for top-1 accuracy and 59.3% for hit@k) and PARALLEL (i.e., >60% for both top-1 accuracy and hit@k). Also, PARALLELwith the separate decoders shows the worst performance even compared to SEQUENTIAL, indicating the importance of capturing the correlation of hypothesis components.

Table 3 presents the results on $Test_2$, a more challenging task subset of $Test_1$, where the ground-truth interpretation is not in the request's n-best interpretation. We see similar trends across the decoders except HIERARCHICAL outperforming the others variants by a much larger margin - 156.2% better in F_1 on slot value, 105.6% better in top-1 accuracy and 145.3% better in hit@k on full hypothesis generation. This result reaffirms the clear value that HIERARCHICAL brings in to the NLU n-best enrichment task. Again, PARALLEL showed the worst performance overall except the slot values' recall and F_1. We conjecture that this is due to the skewness of the live traffic logs toward popular slot values in our conversational AI usage context, thereby leading to the increased amount of false positives in PARALLEL, specifically when the correlation of hypothesis components is not considered.

5.2 Qualitative Analysis

Table 4 shows three examples for qualitative analysis, with each example showing the generated hypotheses from three different decoder architectures. The examples show that both SEQUENTIAL and PARALLEL have the problem of aligning slot keys and values, especially when the sizes of the predicted slot key and slot value are not the same, while HIERARCHICAL handles them better, which is consistent with the overall performance results. For HIERARCHICAL, all slot keys and values are paired and aligned by the key-value cell, and thus no extra filtering or aligning is needed. Even for the first example, which could be considered relatively easy for generation since the input utterance and hypothesis features are very similar to the target hypothesis, both SEQUENTIAL and PARALLEL have trouble with slot information.

The first example would generate and inject an NLU hypothesis that is semantically opposite of the original request (turn on vs. turn off). This opposite hypothesis can still be valuable to identify and/or recover from ASR errors. If it is undesirable, a post-processing step can prevent opposite-meaning hypotheses injection. The second example shows that it tried executing a completely different third-party intent, but it is most likely to be corrected if the hypothesis from the rephrased utterance was executed in the first place. The third example is similar to the second example but the original and rephrased utterances are very different.

[2]We will disclose the exact performance metrics once the internal confidentiality review is complete.

Example 1: the utterance, intent and one slot value are different	
Input utterance	"turn the bedroom light on"
1-best hypothesis	*Intent:* TurnOn, *Slots:* [Appliance: "bedroom light", Action: "turn on", DeviceName: "bedroom light"]
Rephrased utterance	"turn the bedroom light off"
Executed hypothesis	*Intent:* TurnOff, *Slots:* [Appliance: "bedroom light", Action: "turn off", DeviceName: "bedroom light"]
SEQUENTIAL	TurnOff, ⟨EOI⟩, Appliance, Action, DeviceName, ⟨EOK⟩, "bedroom light", ⟨EOV⟩
PARALLEL	*Intent:* TurnOff, *Slot key:* [Appliance, DeviceLocation, DeviceType, DeviceName], *Slot value:* ["bedroom light", "bedroom", "turn off", "christmas tree", "turn on"]
HIERARCHICAL	[TurnOff, [(Appliance, "bedroom light"), (Action, "turn off"), (DeviceName, "bedroom light")]]
Example 2: the utterances are the same but the hypotheses are totally different	
Input utterance	"play christmas songs by kidz bop kids"
1-best hypothesis	*Intent:* ThirdPartySkill, *Slots:* []
Rephrased utterance	"play christmas songs by kidz bop kids"
Executed hypothesis	*Intent:* PlayMusic, *Slots:* [MediaType: "SONG", ArtistName: "kidz bop kids", GenreName: "christmas"]
SEQUENTIAL	PlayMusic, <EOI>, MediaType, GenreName, <EOK>, "christmas", "SONG", <EOV>
PARALLEL	*Intent:* PlayMusic, *Slot key:* [MediaType, ThirdPartySlot, SongName, Time, OnType], *Slot value:* ["SONG", "christmas music", "christmas", "ALARM"]
HIERARCHICAL	[PlayMusic, [(MediaType, "SONG"), (ArtistName, "kidz bop kids"), (GenreName, "christmas")]]
Example 3: the utterances and the hypotheses are different	
Input utterance	"search for bluetooth"
1-best hypothesis	*Intent:* QA, *Slots:* [Question: ""]
Rephrased utterance	"connect to my bluetooth device"
Executed hypothesis	*Intent:* ConnectBluetoothDevice, *Slots:* [Device: "device", ContactName: "device"]
SEQUENTIAL	ConnectBluetoothDevice, ⟨EOI⟩, ⟨EOK⟩, ⟨EOV⟩
PARALLEL	*Intent:* QA, *Slot key:* [], *Slot value:* []
HIERARCHICAL	[ConnectBluetoothDevice, [(Device, "device"), (ContactName, "device")]]

Table 4: Examples of qualitative analysis, showing the generated NLU hypotheses from each Hypgen decoder architecture.

6 Limitations and Future Work

In this paper, we focus on investigating the value of leveraging the inherent structure of a typical NLU semantic interpretation for HypGen, not on obtaining the best performance. We leave extending this work with recent state-of-the-art techniques (e.g., based on Transformer and BERT) for future work. Still, the proposed approach could either inject a new promising interpretations toward more diversity, or if not new, indicate that other non-top interpretations are very promising based on user rephrase behavior. Even with noise, the generated interpretations would still be valuable, and how best to leverage the additional context can be determined and fine-tuned per use case. We also limit the evaluation of the proposed system up to the NLU n-best enrichment step and leave the potential end-to-end production integration impact for future work.

This work can be integrated into the production systems in multiple ways. For instance, we can use HypGen to automatically assign new labels to defective traffic and try to correct them by curating new supervision data to directly improve NLU models. We can perform online exploration by executing the generated interpretations and measure end-to-end user experience impact to determine whether to converge on them. The generated interpretations can be used in a further re-ranking step in downstream components (such as individual domains that often do their own re-ranking on the received NLU n-best information with domain-specific signals). We can also use the generated interpretations for asking clarification questions for disambiguation.

7 Conclusion

We proposed HypGen, a novel generative approach for automatic NLU n-best enrichment in the context of a conversational AI system. To obviate manual annotation needs, we leveraged implicit user feedback for automatic supervision data curation. We designed and compared three different modeling choices for HypGen. The experiments showed that the hierarchical modeling choice largely outperforms the others, showing 145% improvement in full hypotheses hit@k when ground-truth not in NLU n-best. The experiments are based on a specific subset of traffic with a rephrase, which gives us only a preliminary view of our approach's promises. To fully assess our method, we will apply it in the broader traffic and quantify opportunities and performance in a more fine-grained way, by integrating it into the runtime system.

References

Ian Beaver and Abdullah Mueen. 2020. Automated conversation review to surface virtual assistant misunderstandings: Reducing cost and increasing privacy. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(08):13140–13147.

Keping Bi, Choon Hui Teo, Yesh Dattatreya, Vijai Mohan, and W Bruce Croft. 2019. Leverage implicit feedback for context-aware product search. *arXiv preprint arXiv:1909.02065*.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media, Inc.

Praveen Kumar Bodigutla, Aditya Tiwari, Spyros Matsoukas, Josep Valls-Vargas, and Lazaros Polymenakos. 2020. Joint turn and dialogue level user satisfaction estimation on mulit-domain conversations. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 3897–3909.

Jose Camacho-Collados and Mohammad Taher Pilehvar. 2018. From word to sense embeddings: A survey on vector representations of meaning. *Journal of Artificial Intelligence Research*, 63:743–788.

Ali El-Kahky, Xiaohu Liu, Ruhi Sarikaya, Gokhan Tur, Dilek Hakkani-Tur, and Larry Heck. 2014. Extending domain coverage of language understanding systems via intent transfer between domains using knowledge graphs and search query click logs. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4067–4071. IEEE.

Dilek Hakkani-Tür, Frédéric Béchet, Giuseppe Riccardi, and Gokhan Tur. 2006. Beyond asr 1-best: Using word confusion networks in spoken language understanding. *Computer Speech & Language*, 20(4):495–514.

Ruining He and Julian McAuley. 2016. Vbpr: visual bayesian personalized ranking from implicit feedback. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *Advances in neural information processing systems*, pages 2042–2050.

Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM.

Veton Kepuska and Gamal Bohouta. 2018. Next-generation of virtual personal assistants (microsoft cortana, apple siri, amazon alexa and google home). In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 99–103. IEEE.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations, San Diego, California, USA*.

Julia Kiseleva, Kyle Williams, Jiepu Jiang, Ahmed Hassan Awadallah, Aidan C Crook, Imed Zitouni, and Tasos Anastasakos. 2016. Understanding user satisfaction with intelligent assistants. In *Proceedings of the 2016 ACM on Conference on Human Information Interaction and Retrieval*, pages 121–130.

Thomas K Landauer, Peter W Foltz, and Darrell Laham. 1998. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284.

Mingda Li, Weitong Ruan, Xinyue Liu, Luca Soldaini, Wael Hamza, and Chengwei Su. 2020. Improving spoken language understanding by exploiting asr n-best hypotheses. *arXiv preprint arXiv:2001.05284*.

Christopher Manning and Hinrich Schutze. 1999. *Foundations of statistical natural language processing*. MIT press.

Rada Mihalcea, Courtney Corley, Carlo Strapparava, et al. 2006. Corpus-based and knowledge-based measures of text semantic similarity. In *Aaai*, volume 6, pages 775–780.

Jonas Mueller and Aditya Thyagarajan. 2016. Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Deepak Muralidharan, Justine Kao, Xiao Yang, Lin Li, Lavanya Viswanathan, Mubarak Seyed Ibrahim, Kevin Luikens, Stephen Pulman, Ashish Garg, Atish Kothari, et al. 2019. Leveraging user engagement signals for entity labeling in a virtual assistant. *arXiv preprint arXiv:1909.09143*.

Sunghyun Park, Han Li, Ameen Patel, Sidharth Mudgal, Sungjin Lee, Young-Bum Kim, Spyros Matsoukas, and Ruhi Sarikaya. 2021. A scalable framework for learning from implicit user feedback to improve natural language understanding in large-scale conversational ai systems.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543. ACL.

Pragaash Ponnusamy, Alireza Roshan Ghias, Chenlei Guo, and Ruhi Sarikaya. 2019. Feedback-based self-learning in large-scale conversational ai agents. *arXiv preprint arXiv:1911.02557*.

Yi Qi, Qingyun Wu, Hongning Wang, Jie Tang, and Maosong Sun. 2018. Bandit learning with implicit feedback. In *Advances in Neural Information Processing Systems*, pages 7276–7286.

Nils Reimers, Iryna Gurevych, Nils Reimers, Iryna Gurevych, Nandan Thakur, Nils Reimers, Johannes Daxenberger, and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*.

Ruhi Sarikaya. 2017. The technology behind personal digital assistants: An overview of the system architecture and key components. *IEEE Signal Processing Magazine*, 34(1):67–81.

Ivan Srba and Maria Bielikova. 2016. A comprehensive survey and classification of approaches for community question answering. *ACM Transactions on the Web (TWEB)*, 10(3):1–63.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Jason D Williams and Suhrid Balakrishnan. 2009. Estimating probability of correctness for asr n-best lists. In *Proceedings of the SIGDIAL 2009 Conference*, pages 132–135.

Wei-Nan Zhang, Lingzhi Li, Dongyan Cao, and Ting Liu. 2018. Exploring implicit feedback for open domain conversation generation. In *Thirty-Second AAAI Conference on Artificial Intelligence*.