LINGUIST: Language Model Instruction Tuning to Generate Utterances for Intent Classification and Slot Tagging

Anonymous ACL submission

Abstract

We present LINGUIST, a method for generating synthetic data for Intent Classification and Slot Tagging (IC+ST) based on a 5B multilingual seq2seq model fine-tuned on a flexible instruction prompt. On a 10-shot setting for learning a new SNIPS intent, we show absolute improvement of +2.5 points (IC) and +2.8 points (ST) over data upsampling, and combined gains of +4.7 points (IC) and +3.2 points (ST) when combined with Back-Translation. On an internal production dataset for Conversational Agent IC+ST, we show between 7.9% and 25.2% relative improvement compared to an internal baseline across four languages. To the best of our knowledge, we are the first to use instruction fine-tuning of a large scale seq2seq model to generate *slot-labeled* data.

1 Introduction

The machine learning models that power Conversational Agents such as Amazon Alexa, Apple Siri or Google Assistant to peform Semantic Understanding (e.g., Intent Classification and Slot Tagging–IC+ST) on user requests normally rely on large quantities of labeled training data. Such labeled data is plentiful for established customer usage patterns, yet scarce when developing new intents and slots. In those cases, a small number of manually curated examples are used as a "starter" dataset, which can be enhanced by Data Augmentation (DA) methods such as paraphrasing (Prakash et al., 2016), back-translation (Bannard and Callison-Burch, 2005), and slot replacement (Zhang et al., 2020).

One important shortcoming of the existing techniques, is that they excel either at generating novel sentences via paraphrasing or at generating new entities via replacement, but not both. Moreover, paraphrasing and translation techniques are better suited for classification tasks rather than ST since slot alignment between the original sentence and the paraphrased one is often noisy and unreliable. Additionally, while there is substantial breadth and quantity of approaches dealing with DA on English, DA for non-English and multilingual settings remains relatively under-explored.

```
INPUT:
<context>
  Get Weather,
  English Output Language
</context> <example>
  "rain"1 (condition_description),
  "East Tawakoni"2 (city),
  "sep. 25th"3 (time_range)
<br>
  Will it "rain"1 on "sep. 25th"3 in
    "East Tawakoni"2
</example> <example>
  "snowy"1 (condition_description),
  "*"2 (city),
  "*"3 (time_range)
<br>

OUTPUTS:

1 What is the "snowy"1 weather like
    in "Kirkland"2 on "Jan. the 11th"3

2 Is it going to be "snowy"1 at
    "12 o'clock"3 in "Westport"2?

3 What's the "snowy"1 weather like
    in "Marysville"2 "two weeks from now"3

4 What "snowy"1 conditions are expected
    in "Mountain View"2 on "Jan. the 15th"3

5 Is it "snowy"1 in "West Oyster Bay"2
    on "July 5th"3?
```

Figure 1: LINGUIST uses an instruction prompt to generate data with both user-requested slot values ("snowy") and model-generated values ("*"). This model **has not seen any training data for GetWeather intent**: it was trained only on the other 6 SNIPS intents.

Recent research has shown that large pretrained Language Models (LM) are very good in generalization given a handful of examples in the context of zero/few-shot learning (Xue et al., 2021; Zheng et al., 2021). However, deploying such a model in production is impractical due to their latency and memory consumption. In this work, we introduce **L**anguage model **IN**struction tuning to **G**enerate **U**tterances for **I**ntent classification and

Slot **T**agging (**LINGUIST**) as an efficient way to benefit from large LMs in production systems. We utilize an internal 5B parameter seq2seq model by fine-tuning it on an instruction-based task for generating *slot-labeled* data given only the intent name and a small number of in-context examples (see Fig. 1 for the input/output format during training and inference). As slots are generated without needing a second stage of slot alignment, not only can we generate completely novel sentences with the right slot tags but the model is better able to generate sentences in inflecting non-English languages with correct grammar than simple slot replacement. For example, using a different variation of the definite article in French: "*la lumière*" ("the light"), "*les lumières*" ("the lights"), "*le ventilateur*" ("the fan"), and "*l'horloge*" ("the clock"), where simple replacement of the slot value would result in an ungrammatical sentence.

We evaluate our approach in two settings: on a "New-Intent Few-Shot" (NIFS) setting for the public SNIPS dataset where we show absolute improvements of +2.5 points (IC) and +2.8 points (ST), and also a multilingual production dataset, where we achieve a relative Semantic Error Rate (SemER) reduction of 7.9% to 25.2%.

This work demonstrates that by using a large-scale seq2seq model we can: (1) directly produce data with slot labels, thus removing the need for a separate label alignment or projection step, (2) use a single multi-lingual model to generate data in multiple languages, (3) control generation to contain specific slot types, user-supplied slot values, or model-generated slot values, and (4) generate data for new intents and slots without any further fine-tuning, using an instruction prompt.

2 Related Work

2.1 Few-shot Learning using Large-Scale LMs

Recently, Large Generative LMs such as GPT-3 (Brown et al., 2020) are shown to perform well in few-shot settings via "in-context learning", where tasks are formulated as text continuation, and the model learns them without any parameter updates.

Wang et al. (2021) aim to entirely remove the need for human-labeled data, by using a large LM to generate synthetic examples for text classification. In a much more recent work, Wei et al. (2022) demonstrate the use of fine-tuning while maintaining the generic knowledge of the model by providing natural-language instructions in a setting they call *instruction tuning*, where a large number of tasks are presented to a model for fine-tuning in a multitask learning fashion. Wei et al. show that their FLAN model outperforms the similarly-sized GPT-3 by a significant margin. Our work is most alined with the work by Wei et al. (2022) but in the context of data generation.

2.2 Data Augmentation via Paraphrasing

An early and still widely used approach for generating paraphrases is Back-Translation (BT). Bannard and Callison-Burch (2005) use a parallel corpus to extract paraphrases directly from parallel data across languages. Sennrich et al. (2016) and Edunov et al. (2018) use data generated via back-translation to improve a Machine Translation (MT) system. Xie et al. (2020) use a pivot language to create paraphrases in the same language without (unlike Bannard and Callison-Burch (2005)) requiring a parallel corpus–translating English to French, then back to English including n-best lists to create multiple paraphrases.

Other approaches directly target the paraphrasing task, such as Prakash et al. (2016) who train an LSTM-based seq2seq model in a supervised fashion on large paraphrase corpora, or Kumar et al. (2020) who drop nonessential words from an input sentence to create a de-noising data set that can be used to fine-tune a pre-trained model. Kumar et al. (2020) apply a similar approach to ours, however they only use the augmented data for intent classification, while we focus on both intent classification and slot tagging.

2.3 Data to Text Generation

While the above approaches aim to capture general linguistic variations (paraphrases, reformulation), a different thread of research creates synthetic utterances directly from the target annotation, Malandrakis et al. (2019) propose a seq2seq model that learns from interpretation-text pairs seen in training to generate text, with the addition of a controlled variational auto-encoder to improve variability. Jolly et al. (2020) expand on this, exploring different sampling strategies, adding more variety by shuffling slot names, and examining the behavior where a new intent is introduced with limited training data.

The DAGA (Ding et al., 2020) approach generates labeled data for ST using a small one-layer LSTM, however it lacks the controllable mechanism for specific slot labels and values that we

develop for LINGUIST. The CTRL paper (Keskar et al., 2019) proposes natural language "control codes" to instruct the model to generate data in a particular style or domain. Their work does not cover generating structured text such as for ST.

2.4 Token Replacement

The SeqMix (Zhang et al., 2020) approach replaces tokens in an ST task using the nearest neighbor in the embedding space. Dai and Adel (2020) apply local transformations of ST words, such as replacing words with synonyms, or with mentions in other instances of the same label in the corpus. Easy Data Augmentation (Wei and Zou, 2019) similarly applies simple token-level changes including synonym replacement, random insertion, random swap, and random deletion. Their work is only on sequence classification like IC, not sequence labeling like ST.

3 LINGUIST Data Generator Model

Figure 1 shows the format of the instruction prompt that is used in finetuning and inference, based on a pre-trained multi-lingual seq2seq model. It contains three blocks: (1) intent name and output language name, (2) up to 10 slot-annotated examples for the intent, and (3) instructions to determine which slot types and slot values to generate. Each example in the prompt has two components: first the list of slot types and values with numbered quotations, e.g. `"jason mraz"1 (artist)`, then after the separator token
, a formatted version of the fully annotated utterance e.g. `play "jason mraz"1 songs`. The generation instructions use the same slot tagging scheme, e.g. `"weezer"1 (artist)`, to instruct the model to include "weezer" as the artist, and additionally support a wildcard `"*"1 (artist)` instructing the model to fill in an appropriate value.

Labeled examples for the given IC+ST task are formatted into prompts and de-duplicated into a training dataset R. After grouping utterances by "semantic shape"[1] (intent and unordered[2] set of slots), we format an instruction prompt p_i targeting each annotated utterance $t_i \in R$, including in the prompt 10 *other* example utterances $E = \{e_j\}_{j=1}^{10} \in R$ s.t. $\forall j,\ e_j \neq t_i$ and $\text{shape}(e_j) = \text{shape}(t_i)$. To make the generation robust to the number of provided examples, we do not always include all 10 in the prompt, but instead randomly select a value k between 0 and 10, then randomly select k examples from R. If there are only $n < 10$ other utterances available that share the same shape as t_i, then we sample k between 0 and n. We never duplicate utterances in the prompt. Finally, we produce a corpus of training prompts equal in size to the original IC+ST training set.

To account for novel intents and slot types that were not present in training data, we drop out the intent and slot names at a rate of 0.2, replacing the original label name such as `Get Weather` with a random sequence of between 1 and 5 capital letters like `A Q Y`.

In order to generate slot-labeled data, the model must produce some symbols other than just the words of the utterance. In early experiments, we tried a variety of output formats, such as `[artist](jason mraz)`, however found that the model struggled to generate the brackets and label names correctly. After many rounds of experiments, we designed a scheme using quotations and numbers (e.g., `"jason mraz"1`) which we found the model is capable of learning to generate correctly. This format was inspired by natural text examples, e.g. links and references in Wikipedia articles sometimes appear in this format.

To jointly teach the model both to copy user-supplied slot values like `"jason mraz"1 (artist)` and to produce appropriate values for the wildcard `"*"1 (artist)`, we format the training prompts with examples of both. For prompt p_i targeting utterance t_i, we randomly select $d \sim \text{Geo}(0.5)$ $(0 \leq d \leq \#\text{ slots in } t_i)$ slots types and replace their values with `"*"` in the final block of the prompt.

Finally, we apply standard seq2seq fine-tuning of the pre-trained model on the prepared training data, where the prompt text p_i is fed into the encoder, and the decoder is optimized via cross-entropy loss to produce the formatted ground-truth slot-annotated utterance t_i. For more details on the fine-tuning please refer to Appendix A.

In summary, our model is trained to generate utterances with intent and slot labels, conditioned on a handful of examples and a flexible prompt to control the output. We call our Data Generation

[1] For example, `play some [year](1991) [artist](Dave Barker)` and `I want to hear [artist](Steven Harwell) from the [year](thirties)` have the same semantic shape of `PlayMusic` with unordered set of slots `artist` and `year`.

[2] Grouping training data by *unordered* set of slots enables LINGUIST to generate slots in any order it sees fit.

method **LINGUIST**, which stands for **L**anguage model **IN**struction tuning to **G**enerate **U**tterances for **I**ntent classification and **S**lot **T**agging.

4 Experimental Setup

We evaluate LINGUIST by generating training data to augment a few-shot setting for fine-tuning a small encoder model on joint IC+ST. In this section, we describe the datasets and the baselines that we use for evaluating our method.

4.1 Datasets

We present results on two datasets: (1) the public SNIPS dataset (Coucke et al., 2018), and (2) an internal IC+ST dataset.

For each dataset, we construct a **"New-Intent Few-Shot" (NIFS) setting**, shown in Figure 2. Starting with training data $R = \bigcup_{j=1}^{m} D_j$ containing data D_j for m intents $j = 1 \ldots m$, we select an intent $i \in \{1, \ldots, m\}$, and reduce its training data to only a handful of "starter" utterances $S_i \subset D_i$. We then apply various data augmentation techniques on S_i to create augmented data A_i. Finally, we train an IC+ST model using $R'_i = S_i \cup A_i \cup \{D_j\}_{j \neq i}$, i.e. the concatenation of starter and augmented data for intent i with the unmodified data for all other intents.

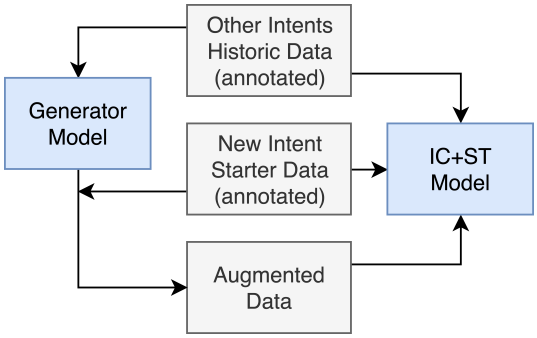


Figure 2: New-Intent Few-Shot (NIFS) Setup.

4.1.1 SNIPS Dataset

The SNIPS dataset (Coucke et al., 2018) is a public IC+ST benchmark consists of 7 intents, each with between 2 and 14 slot types (39 unique slot types in total). It includes around 2k training utterances and 100 validation utterances per intent. In order to avoid overfitting our method on the small validation set, at the beginning of our experiments, we partition the training set into 97% Train and 3% Development. We use our Development set split for iterating on all modeling and data processing decisions, including the hyperparameters for LINGUIST and hyperparameters and selection of best checkpoint for the encoder fine-tuning on IC+SF. Only at the very end of our experiments, we evaluate and report on the Validation set. See Table 1 for count of Train/Dev/Valid utterances.

Intent	Train	Dev.	Valid.
AddToPlaylist	1884	58	100
BookRestaurant	1914	59	100
GetWeather	1940	60	100
PlayMusic	1940	60	100
RateBook	1898	58	100
SearchCreativeWork	1896	58	100
SearchScreeningEvent	1901	58	100
Total	13373	411	700

Table 1: Data counts per intent for SNIPS.

To demonstrate the ability of LINGUIST to generalize to new intent and slot values not present at training time, we construct a **"New-Intent Few-Shot" (NIFS) setting for SNIPS**. Specifically, we create 7 experimental settings as follows: for each intent i of the 7 SNIPS intents, we discard its training data D_i *during fine-tuning*. Then, *during inference*, we randomly select 10 "starter" training examples $S_i \subset D_i$ (ensuring there is at least one example for all slot types related to intent i) to be included in the input prompt.

For each of the 7 intent settings, we create five versions using a different random seed for selecting the 10 starter utterances used during inference.

4.1.2 Internal Dataset

Our internal few-shot IC+ST setting consists of de-duplicated, anonymized, and de-identified customer requests to a large-scale production Conversational Agent. We follow a similar procedure to SNIPS described in Section 4.1.1. The main differences are: (1) we prepare experiments at the level of a *feature*, which may correspond to multiple new intents and/or new slot types or slot values for existing intents, and; (2) the benchmark is multilingual.

There is a large corpus of Historical training data H, containing annotated utterances from user interactions prior to the new features being launched. *The training data does not contain examples of any of the new features.* For each new feature i there is a small dataset S_i, containing between 3 and 429 manually created and annotated "starter" utterances. Then, there is a larger test set T_i consisting of post-launch labeled examples of real customer interactions for the new feature i.

The training set contains several million examples covering hundreds of intents and slot types, and 8 languages: German, English, Spanish, French, Hindi, Italian, Japanese, and Portuguese. The test sets contain hundreds of utterances on five features: CameraControl, ClockSettings, HomeSecurity, Music, and Timers, across four languages: German, Spanish, French, and Japanese.

We train a single LINGUIST model on the historical data H. Then for each feature i, we create prompts from its starter utterances S_i, generate augmented data A_i, and train an IC+ST model M_i on $R'_i = S_i \cup A_i \cup H$.

4.2 Baseline Methods

The **Interpretation-Conditioned Language Model** (ICLM) Jolly et al. (2020) generates unlabeled text conditioned on intent and provided slot values, with a separate label projection step to recover the full slot annotation. ICLM produces alternative carrier phrases only, and cannot generate novel slot values. Our implementation uses a small Transformer architecture with $\sim$ 3.4M parameters, and a simple character-level Levenshtein distance measure to project the slot labels. One advantage of this small model size is low latency when generating data. For SNIPS, we produce 50 outputs per input, then filter and de-duplicate (see Appendix E). For the internal benchmark, we use a similar setting.

We apply **Back-Translation** (Sennrich et al., 2016) (BT) with two separate MT systems. The first uses the open-source Sockeye toolkit (Hieber et al., 2018) and a small (91M parameters) internal Transformer seq2seq model which has been fine-tuned on around 10k utterances of annotated parallel data. We use fast_align (Dyer et al., 2013) to project the slot labels to the generated utterances. We call this system **"BT-Small"**. For SNIPS, we use French as the pivot language, with beam search and $M = 1$ forward en$\rightarrow$fr, then $N = 10$ back-translations fr$\rightarrow$en. We filter to discard noisy outputs (See Appendix F for details) and de-duplicate. For the Internal Benchmark, we use a similar setting, with English as the pivot language.

For a stronger BT baseline, we build a MT system using the very same internal pre-trained multi-lingual 5B seq2seq model which we use for fine-tuning LINGUIST. We fine-tune the 5B model on WMT14[3] jointly on en$\rightarrow$fr and fr$\rightarrow$en using an instruction prompt (simply prefix the input text with `Translate to French:` or `Translate to English:`, respectively) to control the translation direction. We apply SimAlign (Jalili Sabet et al., 2020) to transfer the slot labels to the generated utterance. We use the default `bert-base-multilingual-cased`[4] as the alignment model, and `ArgMax` for matching. For SNIPS, we use French as the pivot language, with $M = 10$ forward and $N = 10$ backward for beam search, thus producing 100 outputs per original sentence, then filter and de-duplicate the outputs (see Appendix G for details). We call this system **"BT-5B"**. Note, we do not include BT-5B for the Internal Benchmark, as it was not available at the time we conducted evaluation.

Catalog Resampling is a simple approach to data augmentation which samples entities from an external catalog for a particular label. For example, if an utterance for `PlayMusic` is `play songs by [artist] (jason mraz)` and there is an external catalog for artist names containing `weezer`, this method could produce an utterance `play songs by [artist] (weezer)`. We only use Catalog Resampling for the Internal Benchmark, as we do not have slot catalogs available for the SNIPS data.

4.3 IC+ST Model Training

For SNIPS, following Chen et al. (2019), we fine-tune a BERT-style model for joint IC+ST. On top of the encoder hidden states, we attach two separate classification heads, one for IC and another for ST. Each head consists of two layers of 256 hidden dimension, with gelu activation, dropout 0.2, and layer norm. The IC head utilizes representation from the first token of the sequence (`[CLS]`), while the ST head utilizes the first subword token of each slot-labeled word.

For our encoder, we use `xlm-roberta-base` (Conneau et al., 2020) (12 layers, 768 hidden dimension), from the HuggingFace (Wolf et al., 2020) implementation. We fine-tune with batch size 128 for 3k updates (i.e. 30 epochs for the full-size data). We freeze the embedding layer; all other parameters are free to update during training. We use Adam with peak learning rate 3e-5, increased linearly from 0 to 600 updates, then decayed linearly to 0 for the remainder of training. To avoid overfitting on the official SNIPS Validation dataset, we

[3] https://huggingface.co/datasets/wmt14

[4] https://huggingface.co/bert-base-multilingual-cased

use our Development split (Section 4.1.1) for early stopping, selecting the checkpoint with best performance on ST. All of our IC+ST fine-tuning runs for SNIPS use identical hyper-parameters, regardless of the data generation method being explored. For each experiment, we train and test 7 different Joint IC+ST models $\{M_i\}_{i=1}^{7}$ in NIFS setting: using a combination of the modified data for intent i, and the unmodified data for all other intents.

For our internal benchmark, we follow a similar procedure to our SNIPS experiments, however we use a smaller internal Transformer-based encoder for fine-tuning on the IC+ST task.

4.4 Metrics

4.4.1 Metrics for SNIPS

We use separate metrics to measure (1) support for the new intent, while (2) not harming the overall customer experience across all intents. For (1), we run the model on a test set containing *only* the new intent. We refer to this as the *Local* Intent Recall (IR), and *Local* ST F1 Score. For F1 score, we ignore the "O" (non-entity) tag. To measure (2), we run the model on the combined test set of all intents together, and call this the *Global* Intent Accuracy (IA) and *Global* ST F1 Score.

When training data is modified for a particular intent i, we expect the Local metrics for i to change a lot, and the Global metrics to be less sensitive. Since this paper focuses on data generation targeting a specific intent or slot, **we are primarily concerned with the Local metrics**.

4.4.2 Metrics for Internal Benchmark

For the internal benchmark, we only evaluate in the *Local* setting. We measure Semantic Error Rate (SemER) (Su et al., 2018) which jointly evaluates the IC and ST performance. Lower SemER indicates improvement to the system. We report relative reduction in SemER, where a negative number indicates improvement.

5 Results

5.1 SNIPS

The main results are presented in Table 2a for Local Intent Recall and Table 2b for Local ST F1 Score.

5.1.1 SNIPS Baselines

As an upper bound for our experiments in the New-Intent Few-Shot (NIFS) setting, we first report the results from training on the full unmodified dataset ("Full" in the tables), showing 99.2 for Local Intent Recall and 96.6 for Local ST F1 Score.

Next, representing the starting point for developing a new intent i, we train the model on the concatenation of the 10 starter utterances S_i for intent i, with all other intents' training data unmodified. ("s10-NoUps" in the tables.) This data reduction severely harms performance of the targeted intent, causing -14.8 points absolute on Local Intent Recall (from 99.2 to 84.4), and -29.8 points absolute on Local ST F1 Score (from 96.6 to 66.8).

Interestingly, simply up-sampling[5] the starter utterances ("s10" in the tables) to recover the original per-class distribution provides a large boost in performance. Specifically, "s10" compared to "s10-NoUps" provides absolute improvement of +3.8 points absolute on Local Intent Recall (from 84.4 to 88.2), and +10.9 points absolute on Local ST F1 Score (from 66.8 to 77.7).

The rest of the columns use a mix of the up-sampled 10 starter utterances, and augmented data derived from them via various methods. We mix the two data sources with portions 0.5/0.5, unless otherwise specified. In all cases, we re-sample the final amount of data for the target intent to match the count in the original unmodified dataset.

We find that ICLM ("s10+ICLM") and BT-Small ("s10+BT-Small") do not improve on Local Intent Recall or Local ST F1 Score compared to "s10". Thus, we **adopt "s10" as our baseline, i.e. using only the 10 starter utterances, up-sampled to recover the original class distribution**.

Compared to "s10", Back-Translation-5B ("s10+BT-5B") shows +1.9 points absolute on Local Intent Recall (from 88.2 to 90.1), and +1.5 points absolute on Local ST F1 Score (from 77.7 to 79.2).

5.1.2 LINGUIST for SNIPS

We train 7 versions of the LINGUIST model as follows: for each intent i of the 7 SNIPS intents, we discard its training data D_i and train a LINGUIST model L_i only on the remaining 6 intents' data $\{D_j\}_{j \neq i}$. Then, given the 10 starter utterances S_i for intent i, we format prompts as described in Section 3, and feed them to LINGUIST model L_i to generate more data G_i. Note, this step **does not require any model parameter updates**.

[5] We implement up-sampling by simply repeating the 10 starter utterances 194 times, to create a file of 1,940 lines, the same as the original full training data for that intent.

Modified Intent / Data	Full	s10-NoUps	s10 (baseline)	s10 +ICLM	s10 +BT-Small	s10 +BT-5B	s10 +LINGUIST	s10 +LINGUIST +BT-5B
AddToPlaylist	100.0	95.6 ±6.8	98.3 ±2.1	97.5 ±2.2	98.3 ±2.4	**99.4** ±0.5	87.7 ±9.0	95.0 ±6.2
BookRestaurant	100.0	91.0 ±3.4	93.5 ±2.4	93.8 ±1.8	92.5 ±1.7	94.6 ±1.1	**95.2** ±3.2	**96.2** ±1.4
GetWeather	100.0	98.8 ±0.4	98.8 ±0.4	99.4 ±0.5	99.6 ±0.5	**99.8** ±0.4	**99.8** ±0.4	**99.8** ±0.4
PlayMusic	99.0	70.8 ±10.1	77.1 ±6.6	79.8 ±8.4	76.5 ±5.8	84.0 ±2.8	**89.1** ±4.2	**90.6** ±2.3
RateBook	100.0	99.0 ±0.0	99.6 ±0.5	**100.0** ±0.0	99.8 ±0.4	99.6 ±0.5	99.8 ±0.4	99.8 ±0.4
SearchCreativeWork	100.0	69.0 ±8.8	76.9 ±9.3	74.2 ±9.1	73.3 ±13.6	79.4 ±11.3	**86.4** ±8.3	**88.8** ±8.5
SearchScreeningEvent	95.2	66.5 ±5.6	73.1 ±8.2	72.1 ±3.7	71.5 ±6.1	73.5 ±10.8	**76.9** ±2.0	**79.8** ±3.9
Average	99.2	84.4 ±2.9	88.2 ±1.9	88.1 ±2.4	87.4 ±2.9	90.1 ±1.6	**90.7** ±2.2	**92.9** ±1.1

(a) SNIPS New-Intent Few-Shot (NIFS) results on **Local Intent Recall**.

Modified Intent / Data	Full	s10-NoUps	s10 (baseline)	s10 +ICLM	s10 +BT-Small	s10 +BT-5B	s10 +LINGUIST	s10 +LINGUIST +BT-5B
AddToPlaylist	94.1	76.8 ±2.9	81.2 ±2.5	78.4 ±2.4	**82.0** ±1.8	81.3 ±1.7	80.0 ±4.0	78.9 ±2.0
BookRestaurant	96.4	71.9 ±2.3	81.3 ±2.1	80.4 ±1.4	81.2 ±1.2	**83.3** ±2.5	81.2 ±2.9	81.6 ±0.9
GetWeather	97.8	74.7 ±3.9	**84.9** ±5.4	82.9 ±4.8	82.3 ±4.5	84.0 ±2.8	81.6 ±3.1	84.2 ±3.4
PlayMusic	91.7	42.0 ±4.3	59.2 ±2.2	58.0 ±3.4	56.1 ±3.1	65.4 ±4.2	**66.8** ±4.1	**67.1** ±2.7
RateBook	99.7	89.4 ±1.5	95.0 ±0.9	**95.4** ±0.8	93.5 ±3.3	93.6 ±1.5	95.3 ±1.0	95.3 ±1.0
SearchCreativeWork	100.0	56.2 ±10.5	70.9 ±11.3	67.6 ±9.8	68.9 ±11.1	72.9 ±12.1	**81.0** ±9.1	80.4 ±9.6
SearchScreeningEvent	96.6	56.4 ±7.4	71.6 ±3.6	72.8 ±4.6	69.8 ±4.6	74.2 ±3.6	**77.3** ±3.3	**79.1** ±4.4
Average	96.6	66.8 ±2.2	77.7 ±2.0	76.5 ±2.1	76.3 ±2.5	79.2 ±2.8	**80.5** ±2.3	**80.9** ±1.3

(b) SNIPS New-Intent Few-Shot (NIFS) results on **Local ST F1 Score**.

Table 2: Our main results on SNIPS Validation set (Section 4.1.1). For each cell (i, j), we train a joint IC+ST encoder on the combination of data from intent i modified according to strategy j, and all other intents' data unmodified. The first three columns are: "Full" is trained on the full dataset without any modifications; for "s10-NoUps", the data for intent i is reduced to only 10 "starter" examples, and are *Not Up-sampled*; for **"s10", the baseline**, the starter utterances are up-sampled to N_i, the original data size for intent i. For the remaining columns, the up-sampled starter utterances for intent i are mixed with augmented data derived from them using a particular method, which is re-sampled to N_i in size. "st10+ICLM" uses ICLM, "s10+BT-Small" uses the Small Back-Translation system, "s10+BT-5B" uses our Back-Translation with the internal 5B seq2seq model, "s10+LINGUIST" uses data generated by our LINGUIST method. Finally "s10+LINGUIST+BT-5B" uses data from both LINGUIST and BT-5B. We bold the mean for the best single method, and for the final column when it is best. All experiments are run across the five random seeds, and we report mean ± standard deviation.

Utilizing the ability of LINGUIST to both copy slot values and produce novel values, we format multiple prompt versions p_{ik} from each of the starter utterances s_i. The first version instructs LINGUIST to copy all the slot values, producing new carrier phrases. Then, for each slot type k, we create a new version of the prompt replacing the value for k with the wildcard `"*"`, instructing LINGUIST to produce a new value for the slot, while copying the other slot values as they are, and generating a suitable carrier phrase.[6] We use *top_k* sampling with $k = 50$ and temperature 0.3 to generate 100 utterances per prompt.

After filtering the generated data (see Appendix H for details), we mix the up-sampled 10 starter utterances with the LINGUIST-generated data, We explore three settings for the mixing weights, namely 0.3/0.7, 0.5/0.5, and 0.7/0.3, and select 0.5/0.5 according to best Local IR and Local ST F1 Score on the validation set on just one of the seed runs. We use identical settings for LINGUIST fine-tuning and generation across all 35 runs (7 intents times 5 random seeds) for the SNIPS-NIFS benchmark.

Finally, following the setting of the other baselines, we fine-tune `xlm-roberta-base` on the concatenation of the augmented and mixed data for intent i with the original data for all other intents: $R'_i = S_i \cup A_i \cup \{D_j\}_{j \neq i}$.

Compared to the baseline of up-sampled starter utterances ("s10"), LINGUIST improves by **+2.5 points absolute on Local Intent Recall** (from 88.2 to 90.7), and **+2.8 points absolute on Local ST F1 Score** (from 77.7 to 80.5).

Finally, we demonstrate that the improvements

[6] For example, for a `GetWeather` prompt containing `"new york"1 (state), "noon"2 (time_range)` we prepare three versions: one as-is, one with `"*"1 (state), "noon"2 (time_range)`, and one with `"new york"1 (state), "*"2 (time_range)`.

in Local metrics for the new intent do not cause harm to the overall system, and in fact provide a small improvement. As shown in Table 4a and Table 4b (Appendix D), "s10+LINGUIST" improves upon the baseline of "s10" by +0.3 points on both Global Intent Accuracy and Global ST F1 Score.

5.1.3 LINGUIST and Back-Translation

As our final experiment, we combine the up-sampled starter utterances with data from both LINGUIST and BT-5B ("s10+LINGUIST+BT-5B"). The mixing ratio for the three datasets is 0.2/0.4/0.4. We find that the methods are synergistic, and compared to LINGUIST alone, adding generated data from BT-5B adds +2.2 points absolute on Local Intent Recall (from 90.7 to 92.2), and +0.4 points absolute on Local ST F1 Score (from 80.5 to 80.9). The final difference between the baseline of "s10" and the candidate system "s10+LINGUIST+BT-5B" is **+4.7 points absolute on Local Intent Recall** (from 88.2 to 92.9), and **+3.2 points absolute on Local ST F1 Score** (from 77.7 to 80.9).

5.2 Internal Dataset

5.2.1 Internal Benchmark Baseline

We train a separate IC+ST Encoder model M_i for each feature i. The training data for feature i is a mix of the Historical data H, up-sampled starter utterances for the feature S_i, along with augmented utterances A_i produced from S_i via Catalog Resampling, ICLM, and BT-Small. For each feature i, we evaluate the IC+ST model M_i on the test set T_i for that feature, and report the Local SemER.

5.2.2 LINGUIST for Internal Benchmark

We fine-tune a single LINGUIST model on instruction prompts formatted from the full historical dataset H described in Section 4.1.2. We note again that the training data does not contain examples of the new features.

Then, for each feature, i, we follow a similar procedure described in Section 5.1.2: we format prompts from the starter utterances S_i and apply LINGUIST to generate more data G_i. For generation, we use *top_k* sampling with $k = 20$ and temperature 1.0 to produce 20 outputs per prompt.[7] using the same settings for all features i.

[7]We observe that when LINGUIST is trained on a much larger dataset compared to SNIPS, it produces less noisy outputs. Thus, we allow a higher temperature of 1.0 compared to SNIPS setting where we use temperature 0.3.

Finally, we follow the same data mixing, training, and testing procedure for each feature i as in the baseline (Section 5.2.1). As shown in Table 3, LINGUIST results in 7.9% to 25.2% SemER reduction across four languages compared to the baseline system of combined Catalog Resampling, ICLM, and BT-Small.

Feature/Lang	de	es	fr	ja
CameraControl	-	-33.3%	-	-
ClockSettings	-1.6%	-12.3%	+1.8%	-
HomeSecurity	-	-	-	-31.5%
Music	-36.8%	-	-30.6%	-12.3%
Timers	-27.5%	-15.4%	-20.0%	-
Average	-11.8%	-20.8%	-7.9%	-25.2%

Table 3: Results on Internal Dataset. Each number is relative reduction in SemER compared to the baseline system of combined Catalog Resampling, BT, and ICLM. A negative number indicates improvement.

6 Conclusion and Future Work

In this work, we introduced an efficient and flexible method to capitalize on the value of large LMs by utilizing them for generating synthetic data with intent and slot tags. We showed that our method, called LINGUIST, can be used to generate new data while keeping the slots of interest and also generate data for completely new intent and slot values. We compared our method to paraphrasing and slot replacement approaches and showed a significant improvement over these methods both on public and internal datasets.

In future work, we will explore methods to improve the generated data from LINGUIST. Five research directions are of particular interest: (1) training a separate classifier to select the highest quality and most relevant outputs, (2) applying techniques during training to discourage incorrect outputs, e.g. Welleck et al. (2019) and Li et al. (2020), (3) discriminative fine-tuning with human labels of quality of outputs, as explored in the very recent LaMDA paper (Thoppilan et al., 2022), (4) reinforcement Learning with a human-in-the-loop reward signal, and (5) working to extend LINGUIST to generate data for other structured prediction tasks such as semantic parsing and text to SQL.

References

Colin Bannard and Chris Callison-Burch. 2005. Paraphrasing with bilingual parallel corpora. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 597–604, Ann Arbor, Michigan. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Qian Chen, Zhu Zhuo, and Wen Wang. 2019. Bert for joint intent classification and slot filling.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

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.

Xiang Dai and Heike Adel. 2020. An analysis of simple data augmentation for named entity recognition. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3861–3867, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Bosheng Ding, Linlin Liu, Lidong Bing, Canasai Kruengkrai, Thien Hai Nguyen, Shafiq Joty, Luo Si, and Chunyan Miao. 2020. DAGA: Data augmentation with a generation approach for low-resource tagging tasks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6045–6057, Online. Association for Computational Linguistics.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. 2013. A simple, fast, and effective reparameterization of IBM model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648, Atlanta, Georgia. Association for Computational Linguistics.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500, Brussels, Belgium. Association for Computational Linguistics.

Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. 2018. The sockeye neural machine translation toolkit at AMTA 2018. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*, pages 200–207, Boston, MA. Association for Machine Translation in the Americas.

Masoud Jalili Sabet, Philipp Dufter, François Yvon, and Hinrich Schütze. 2020. SimAlign: High quality word alignments without parallel training data using static and contextualized embeddings. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 1627–1643, Online. Association for Computational Linguistics.

Shailza Jolly, Tobias Falke, Caglar Tirkaz, and Daniil Sorokin. 2020. Data-efficient paraphrase generation to bootstrap intent classification and slot labeling for new features in task-oriented dialog systems. In *Proceedings of the 28th International Conference on Computational Linguistics: Industry Track*, pages 10–20, Online. International Committee on Computational Linguistics.

Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation.

Varun Kumar, Ashutosh Choudhary, and Eunah Cho. 2020. Data augmentation using pre-trained transformer models. In *Proceedings of the 2nd Workshop on Life-long Learning for Spoken Language Systems*, pages 18–26, Suzhou, China. Association for Computational Linguistics.

Margaret Li, Stephen Roller, Ilia Kulikov, Sean Welleck, Y-Lan Boureau, Kyunghyun Cho, and Jason Weston. 2020. Don't say that! making inconsistent dialogue unlikely with unlikelihood training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4715–4728, Online. Association for Computational Linguistics.

Nikolaos Malandrakis, Minmin Shen, Anuj Goyal, Shuyang Gao, Abhishek Sethi, and Angeliki Metallinou. 2019. Controlled text generation for data augmentation in intelligent artificial agents. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 90–98, Hong Kong. Association for Computational Linguistics.

Aaditya Prakash, Sadid A. Hasan, Kathy Lee, Vivek Datla, Ashequl Qadir, Joey Liu, and Oladimeji Farri. 2016. Neural paraphrase generation with stacked residual LSTM networks. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Paper s*, pages 2923–2934, Osaka, Japan. The COLING 2016 Organizing Committee.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Chengwei Su, Rahul Gupta, Shankar Ananthakrishnan, and Spyridon Matsoukas. 2018. A re-ranker scheme for integrating large scale nlu models. *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 670–676.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Vincent Zhao, Yanqi Zhou, Chung-Ching Chang, Igor Krivokon, Will Rusch, Marc Pickett, Pranesh Srinivasan, Laichee Man, Kathleen Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Soraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravi Rajakumar, Alena Butryna, Matthew Lamm, Viktoriya Kuzmina, Joe Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Aguera-Arcas, Claire Cui, Marian Croak, Ed Chi, and Quoc Le. 2022. Lamda: Language models for dialog applications.

Zirui Wang, Adams Wei Yu, Orhan Firat, and Yuan Cao. 2021. Towards zero-label language learning. *CoRR*, abs/2109.09193.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

Jason Wei and Kai Zou. 2019. EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6382–6388, Hong Kong, China. Association for Computational Linguistics.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. 2019. Neural text generation with unlikelihood training.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. 2020. Unsupervised data augmentation for consistency training. In *Advances in Neural Information Processing Systems*, volume 33, pages 6256–6268. Curran Associates, Inc.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Rongzhi Zhang, Yue Yu, and Chao Zhang. 2020. SeqMix: Augmenting active sequence labeling via sequence mixup. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8566–8579, Online. Association for Computational Linguistics.

Yanan Zheng, Jing Zhou, Yujie Qian, Ming Ding, Jian Li, Ruslan Salakhutdinov, Jie Tang, Sebastian Ruder, and Zhilin Yang. 2021. Fewnlu: Benchmarking state-of-the-art methods for few-shot natural language understanding.

A Checkpoint Selection

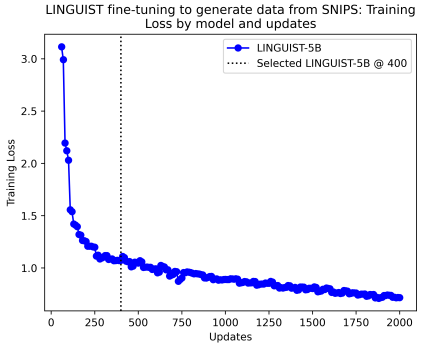
We find that selecting the right checkpoint is crucial in order for LINGUIST to generalize well to new Intent and Slot labels. We observe a "sweet spot" where the model is trained enough to learn the prompt format, yet not too much that it would memorize and overfit to the intents and slots it sees during training. See plots showing metrics across updates: Figure A1 (a), left, for Training Loss, (b) right for Development Perplexity; Figure A2 (a), left, for Development Token Accuracy, and (b), right, for Development Sequence Exact Match.

During initial experiments, we found standard Development metrics like perplexity and accuracy to be unreliable indicators of downstream generation and fine-tuning success. We thus explored manually selecting the checkpoint as follows.

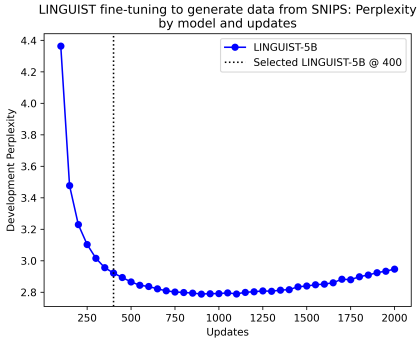
The SNIPS training data has around 13k samples, and we use batch size 512, so each epoch is only about 25 updates. We train for 20 epochs (so around 500 updates) with a constant small learning rate of 5e-7, warmed up over the first four epochs. We shuffle the data with a different seed each epoch. Every 50 updates (so around every 2 epochs), we evaluate metrics on the Development set, and save a checkpoint. We use the Train/Development 97%/3% split described in Section 4.1.1, such that we have around 400 Development samples in total which were not seen in training. We measure Perplexity, Token Accuracy, and Full Sequence Exact Match Accuracy on the Development set. Note that these metrics can be computed with a simple forward pass of the full sequence, i.e. we do not need to run generation.

We note that training task is highly ambiguous. If you asked a human to fill in the blank for `play songs by "*"1 (artist_name)`, it is very unlikely that they would select `jason mraz`, as opposed to some other music artist, without any further information. Note, this is in part because we are filling in entity words, as opposed to random masking done when training MLM models, where the prediction task could be `play songs [MASK] jason mraz` which has significantly less ambiguity, and the ground truth is quite likely to be `by`.

As shown in A2 (b), right, Development Sequence Exact Match never exceeds 3%, so it is likely not informative to select the checkpoint. Perplexity (Fig. A1 (b), right) and Token Accuracy (Fig. A2 (a), left) are both more granular, so we focus on them when selecting the checkpoint.



(a) Training Loss

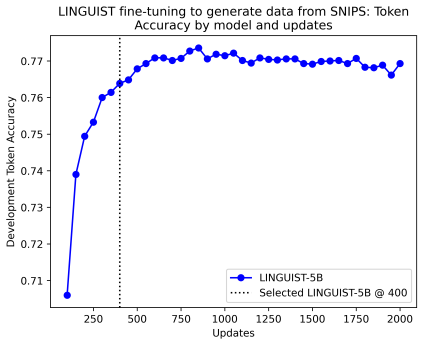


(b) Development Perplexity

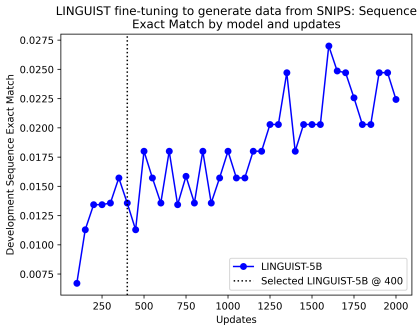
Figure A1: LINGUIST-5B learning to generate SNIPS data: training loss and Development perplexity across model updates.

We manually select the checkpoint based on high (but not too high) Token Accuracy, and manual inspection of generated outputs. For checkpoints at 100, 200, 300, and 400, we perform generation on a few of the held-out intent runs, and choose the best checkpoint according to two criteria: (1) the outputs respect the prompt instructions, e.g., they use the correctly formatted quotations and numbers on the entities, and (2) the outputs are semantically valid according to the prompt intent and slot types.

For example, with held-out intent `AddToPlaylist`, we find that the checkpoint at 50 updates does not meet criteria (1) or (2); after 100 updates, the model meets both criteria (1) and (2), correctly producing semantically valid utterances such as `Please add this "artist"2 to the "Dubstep"1 playlist`. However, at later checkpoints, the decoder seems to overfit to the related "PlayMusic"



(a) Development Token Accuracy



(b) Development Sequence Exact Match

Figure A2: LINGUIST-5B learning to generate SNIPS data: Development token accuracy and sequence accuracy across model updates.

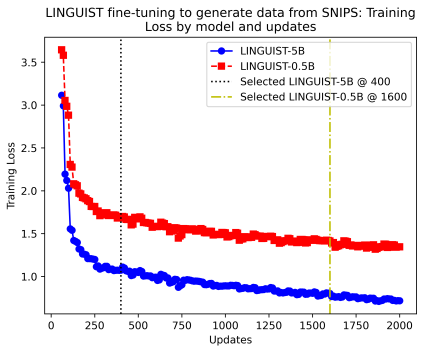
training data, and produces utterances like `Play "Cena Elegante"1 "song"2` which we find hurts both IC and ST when added into the encoder fine-tuning data.

For simplicity, we do not tune the checkpoint selection separately for each held-out intent. Instead, we select the checkpoints at 400 updates for all runs, as that performed best overall.

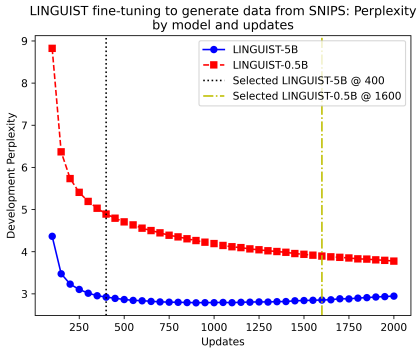
Note that since we train in the New-Intent *Few-Shot* (NIFS) setting described in Section 4.1, the Development set does not contain any new intents compared to the Training set for the LINGUIST model. Selecting based on best Development metrics in this setting could mislead when we are ultimately interested in optimizing for handling new intents well. While it might be interesting to plot token accuracy on the held-out intent itself, (either using the 10 starter utterances, or the original full Validation set) we choose not to run that experiment, as it would pollute any future experiments, since the held-out new intent's Validation set would no longer be blind. As future work, we would like to explore improved methods for checkpoint selection.

B Impact of Model Size

To measure the impact of model size, we also train a 10x smaller LINGUIST-0.5B model using an internal seq2seq model with the same pre-training and fine-tuning setup as our 5B. We plot the Training Loss (Fig. A3 (a), left), Development Perplexity (Fig. A3 (b), right), Development Token Accuracy (Fig. A4 (a), left), and Development Sequence Exact Match (Fig. A4 (b), right) comparing the two models. For Token Accuracy, using a smaller model would cause a drop in 5 points absolute (from 0.774 to 0.724). We also show an example in Fig. A5.

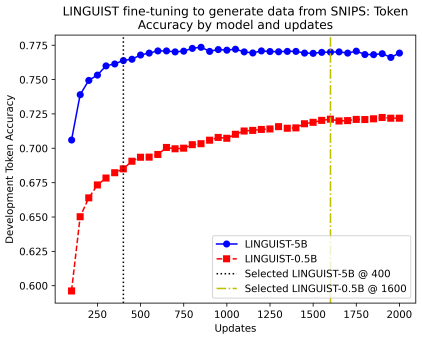


(a) Training Loss

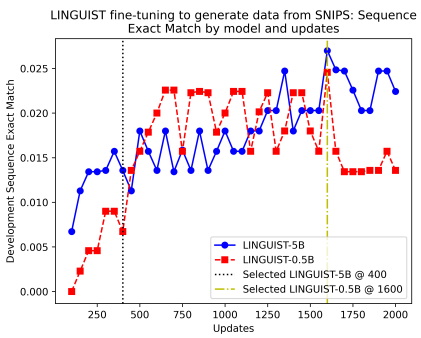


(b) Development Perplexity

Figure A3: Comparing LINGUIST-5B and LINGUIST-0.5B: Train loss and Development perplexity across model updates.



(a) Development Token Accuracy



(b) Development Sequence Exact Match

Figure A4: Comparing LINGUIST-5B and LINGUIST-0.5B: Development token accuracy and sequence accuracy across model updates.

```
INPUT:
<context>
  Get Weather,
  English Output Language
</context> <example>
  "rain"1 (condition_description),
  "East Tawakoni"2 (city),
  "sep. 25th"3 (time_range)
<br>
  Will it "rain"1 on "sep. 25th"3 in
    "East Tawakoni"2
</example> <example>
  "snowy"1 (condition_description),
  "*"2 (city),
  "*"3 (time_range)
<br>

OUTPUTS:

1 What is the "snowy"1 weather like
  in "Kirkland"2 on "Jan. the 11th"3

2 Is it going to be "snowy"1 at
  "12 o'clock"3 in "Westport"2?

3 What's the "snowy"1 weather like
  in "Marysville"2 "two weeks from now"3

4 What "snowy"1 conditions are expected
  in "Mountain View"2 on "Jan. the 15th"3

5 Is it "snowy"1 in "West Oyster Bay"2
  on "July 5th"3?
```

(a) Generated utterances from LINGUIST-5B.

```
INPUT:
<context>
  Get Weather,
  English Output Language
</context> <example>
  "rain"1 (condition_description),
  "East Tawakoni"2 (city),
  "sep. 25th"3 (time_range)
<br>
  Will it "rain"1 on "sep. 25th"3 in
    "East Tawakoni"2
</example> <example>
  "snowy"1 (condition_description),
  "*"2 (city),
  "*"3 (time_range)
<br>

OUTPUTS:

1 Will it get "snowy"1 on
  "jul. 8"3 in "The Point"2

2 Will it be "snowy"1 when it's
  "in the area"3 with the forecast for "now"2

3 Will it be "snowy"1 in
  "Okiwa"2 "may"3

4 Can you get me a forecast for "snowy"1
  conditions at "Tlapa"3 in "Denmark"2

5 Will it be "snowy"1 in
  "Finland"2 on "Sep. 2nd"3
```

(b) Generated utterances from LINGUIST-0.5B.

Figure A5: Generated outputs from LINGUIST-5B (left, (a)) and LINGUIST-0.5B (right, (b)). Crucially, both models **did not see any GetWeather data during training**, as they were trained only on the other 6 SNIPS intents. We observe that while LINGUIST-5B produces primarily high-quality outputs, LINGUIST-0.5B more frequently produces outputs with incorrect slot fills, e.g. `"in the area"` and `"Tlapa"` for `time_range`, or `"Finland"`, `"Denmark"`, and `"now"` for `city`. Furthermore, LINGUIST-0.5B tends to produce less diverse carrier phrases than LINGUIST-5B, which we note tends to remain the case even when we try a larger *top_k* or higher temperature values.

C Sample Model Outputs

We show examples of a LINGUIST model (trained on all 7 SNIPS intents for 400 updates) generating for novel intents and slots not seen during training.

```
INPUT:
<context>
  Draft Player,
  English Output Language
</context> <example>
  "Patrice Bergeron"1 (player),
  "Boston Bruins"2 (hockey_team_name)
<br>
  Assign "Patrice Bergeron"1 to the "Boston Bruins"2
</example> <example>
  "Brad Marchand"1 (player),
  "*"2 (hockey_team_name)
<br>

OUTPUTS:

1 Draft "Brad Marchand"1 for the "Chicago Blackhawks"2

2 Add "Brad Marchand"1 to the "Montreal Canadiens"2 roster

3 Can you put "Brad Marchand"1 on the "Florida Panthers"2?

4 The "San Jose Sharks"2 need "Brad Marchand"1.

5 I want "Brad Marchand"1 to play for the "Philadelphia Flyers"2.
```

Figure A6: The model correctly copies the requested slot `Brad Marchand`, and fills in appropriate values for `hockey_team_name` and uses a variety of carrier phrases.

```
INPUT:
<context>
  Music Theory Get Accidentals,
  English Output Language
</context> <example>
  "d. major"1 (key),
  "sharp"2 (accidental_type)
<br>
  what notes are "sharp"2 in "d. major"1
</example> <example>
  "c. sharp major"1 (key),
  "sharp"2 (accidental_type)
<br>
  tell me which notes are "sharp"2 in "c. sharp major"1
</example> <example>
  "*"1 (key),
  "flat"2 (accidental_type)
<br>

OUTPUTS:

1 Can you tell me which "flat"2 notes are in "fa"1

2 What "flat"2 notes are there in the key of "e"1

3 What are the "flat"2 notes in "si"1

4 find "flat"2 notes in "e flat"1

5 I want to know which notes are "flat"2 in "C"1
```

Figure A7: The model correctly copies the requested slot `flat`, and fills in appropriate values for `key` and uses a variety of carrier phrases.

D Results on Global Metrics

Modified Intent / Data	Full	s10-NoUps	s10 (baseline)	s10 +ICLM	s10 +BT-Small	s10 +BT-5B	s10 +LINGUIST	s10 +LINGUIST +BT-5B
AddToPlaylist	99.1	98.7 ±1.0	98.9 ±0.3	98.9 ±0.3	98.9 ±0.3	99.1 ±0.1	97.4 ±1.3	98.5 ±0.8
BookRestaurant	99.1	97.7 ±0.4	98.2 ±0.3	98.2 ±0.2	97.9 ±0.2	98.2 ±0.1	98.3 ±0.6	98.5 ±0.3
GetWeather	99.1	98.9 ±0.1	98.9 ±0.1	98.9 ±0.2	99.0 ±0.0	99.0 ±0.1	99.0 ±0.1	99.1 ±0.2
PlayMusic	99.1	95.3 ±1.5	96.1 ±1.0	96.4 ±1.3	96.0 ±0.8	97.0 ±0.4	97.8 ±0.6	98.0 ±0.4
RateBook	99.1	99.0 ±0.1	98.9 ±0.1	99.0 ±0.1	99.0 ±0.1	99.0 ±0.1	99.0 ±0.2	99.0 ±0.2
SearchCreativeWork	99.1	95.2 ±1.1	96.5 ±1.2	96.0 ±1.2	95.8 ±1.8	96.7 ±1.4	97.6 ±1.2	98.0 ±1.2
SearchScreeningEvent	99.1	95.3 ±0.8	96.3 ±1.1	95.9 ±0.5	95.8 ±0.8	96.2 ±1.5	96.6 ±0.4	97.1 ±0.6
Average	99.1	97.2 ±0.4	97.7 ±0.2	97.6 ±0.4	97.5 ±0.3	97.9 ±0.2	98.0 ±0.3	98.3 ±0.2

(a) SNIPS New-Intent few-shot results on **Global Intent Accuracy**.

Modified Intent / Data	Full	s10-NoUps	s10 (baseline)	s10 +ICLM	s10 +BT-Small	s10 +BT-5B	s10 +LINGUIST	s10 +LINGUIST +BT-5B
AddToPlaylist	96.7	94.0 ±0.4	94.2 ±0.5	94.2 ±0.4	94.9 ±0.4	94.5 ±0.3	94.1 ±0.5	94.2 ±0.4
BookRestaurant	96.7	92.7 ±0.4	94.1 ±0.6	94.1 ±0.3	94.2 ±0.3	94.5 ±0.3	94.4 ±0.6	94.1 ±0.3
GetWeather	96.7	93.8 ±0.5	94.9 ±0.7	94.7 ±0.6	94.6 ±0.4	95.0 ±0.3	94.4 ±0.5	94.8 ±0.6
PlayMusic	96.7	91.3 ±0.4	92.9 ±0.3	93.1 ±0.6	92.8 ±0.2	93.8 ±0.4	94.0 ±0.5	94.2 ±0.3
RateBook	96.7	95.0 ±0.3	95.8 ±0.4	95.8 ±0.3	95.5 ±0.6	95.5 ±0.4	95.8 ±0.3	95.8 ±0.3
SearchCreativeWork	96.7	92.9 ±1.0	94.2 ±0.9	93.9 ±1.0	94.2 ±1.0	94.1 ±1.1	95.3 ±1.0	95.2 ±0.9
SearchScreeningEvent	96.7	92.3 ±0.8	94.0 ±0.6	94.0 ±0.6	93.5 ±0.6	94.2 ±0.4	94.6 ±0.5	94.8 ±0.7
Average	96.7	93.1 ±0.2	94.3 ±0.2	94.2 ±0.2	94.3 ±0.3	94.5 ±0.3	94.6 ±0.2	94.7 ±0.1

(b) SNIPS New-Intent few-shot results on **Global ST F1 Score**.

Table 4: Our results on SNIPS for the Global metrics, showing that the gains for Local metrics shown in Tables 2a and 2b do not cause harm to the system overall. See Section 5.1 for details.

E Filtering ICLM Outputs

We discard any outputs containing the `<unk>` token, which happens less than 1% of the time. The number of outputs (after de-duplication) are reported in Table 5.

Modified Intent	Num outputs
AddToPlaylist	296
BookRestaurant	347
GetWeather	322
PlayMusic	255
RateBook	288
SearchCreativeWork	295
SearchScreeningEvent	273
Average	297

Table 5: The number of filtered and de-duplicated outputs from ICLM per intent. All numbers are averaged across the five random seeds.

F Filtering BT-Small Outputs

The small model has a fair amount of noise in its outputs, so we heuristically filter them, discarding any which contain repeated bigrams such as `play the song halo the song` and/or any trigram of the same word such as `of of of`. Success rate and number of outputs (after de-duplication) are reported in Table 6.

Modified Intent	SuccessRate	NumOutputs	AvgNumSlots
AddToPlaylist	70.2	64	2.7
BookRestaurant	72.8	73	3.2
GetWeather	60.4	60	2.3
PlayMusic	53.6	52	2.2
RateBook	70.8	71	3.8
SearchCreativeWork	41.6	42	1.8
SearchScreeningEvent	69.6	70	2.2
Average	62.7	62	2.6

Table 6: For each intent, the Success Rate of Back-Translation with the Small model, and Number of Generated Outputs, both averaged across the five random seeds. For reference, we also show the Average Number of Slots in the training data per intent.

G Filtering BT-5B outputs

The Back-Translated text with the 5B model is significantly cleaner than with the smaller model, so we do not apply any filtering on the output text itself. We do heuristically discard any outputs where we suspect the augmented utterance is missing a slot. Specifically, SimAlign in ArgMax mode only returns alignments across words that have mutual argmax between source and target. For any source word that is an entity tag (i.e., not "O"), if it is not aligned to an output word, then we consider the output invalid. For example, an input like `rate this book 5 out of 6` with a Back-Translated output `give this book a rating of 5` would typically have no output word aligned to the source word "6" (`best_rating` slot label), so the output would be discarded.

Success rate and number of outputs (after de-duplication) for BT-5B are reported in Table 7.

Modified Intent	SuccessRate	NumOutputs	AvgNumSlots
AddToPlaylist	66.2	411	2.7
BookRestaurant	82.8	423	3.2
GetWeather	72.0	311	2.3
PlayMusic	89.0	455	2.2
RateBook	79.2	478	3.8
SearchCreativeWork	85.5	451	1.8
SearchScreeningEvent	72.0	330	2.2
Average	78.1	408	2.6

Table 7: For each intent, the Success Rate of Back-Translation with the 5B model, and the number of outputs, both averaged across the five random seeds. For reference, we also show the Average Number of Slots in the training data per intent.

H Filtering LINGUIST Outputs

We apply heuristic filtering by discarding outputs which meet any of the following criteria: (1) copy one of the examples from the prompt verbatim; (2) fail to follow the prompt instructions, by not copying the instructed slot value or by producing repeated, missing, extra, or malformed slot-tag numbers; (3) produce the literal wildcard instruction `"*"`; or (4) produce a punctuation character in the set of {_<>[]()}{};}.[8]

In Table 8, we report the Success Rate as the portion of generated utterances which remain after filtering, and show the total number of generated utterances per intent. We observe a trend that success rate is generally lower when the prompt contains more slots, which is intuitive as the generation task is more challenging and has more chances to make a mistake. The success rates vary significantly by intent from 28.0 for RateBook to 92.9 for SearchCreativeWork, with an average of 67.5 across the 7 intents.

[8]These characters do not appear in the utterance text of any of the original training data, so are considered to be generation mistakes.

Modified Intent	Success Rate	#Outputs	Average #Slots
AddToPlaylist	65.5	1037	2.7
BookRestaurant	55.6	1484	3.2
GetWeather	79.5	1014	2.3
PlayMusic	82.5	956	2.2
RateBook	28.0	497	3.8
SearchCreativeWork	92.9	1290	1.8
SearchScreeningEvent	68.3	967	2.2
Average	67.5	1035	2.6

Table 8: For each intent, the Success Rate of Generation, and Number of Generated Outputs, both averaged across the five random seeds. For reference, we also show the Average Number of Slots in the training data per intent.

I SemER Metric

For the internal IC+ST benchmark (Sections 4.1.2, 4.4.2, and 5.2.2), we report on Semantic Error Rate (SemER) (Su et al., 2018) which jointly evaluates Intent Classification and Slot Filling. SemER is defined as follows: comparing a reference of tokens and their accompanying labels, count each of of these operations: (1) Correct slots, where the slot name and slot value is correctly identified, (2) Deletion errors, where the slot name is present in the reference but not in the hypothesis, (3) Insertion errors, where extraneous slot names are included in the hypothesis, (4) Substitution errors, where slot names from the hypothesis are included but with an incorrect slot value. Intent classification errors are substitution errors. Then, apply Equation 1 to compute the SemER.

$$\text{SemER} = \frac{\text{\# Del} + \text{\# Ins} + \text{\# Sub}}{\text{\# Cor} + \text{\# Del} + \text{\# Sub}} \quad (1)$$