*INTERSPEECH 2021: Show & Tell Contribution*
30 August – 3 September, 2021, Brno, Czechia

# NeMo (Inverse) Text Normalization: From Development To Production

*Yang Zhang, Evelina Bakhturina, Boris Ginsburg*

NVIDIA, Santa Clara, USA
{yangzhang, ebakhturina, bginsburg}@nvidia.com

## Abstract

We introduce the NeMo Text Processing (NTP) toolkit - an open-source Python library for text normalization (TN) and inverse text normalization (ITN) based on weighted finite-state transducers (WFSTs). The English grammars provided within NTP can be seamlessly deployed to the C++ Sparrowhawk framework for production.

**Index Terms**: text normalization, inverse text normalization, WFST, Python, open-source toolkit

## 1. Introduction

Text normalization (TN) converts written text to spoken form and is a part of the text-to-speech (TTS) preprocessing pipeline. Inverse text normalization (ITN) does the opposite and converts spoken-domain automatic speech recognition (ASR) output into written-domain text to improve the readability of the ASR output. For example, ITN would make the following conversion: *"on may third i paid one hundred and twenty three dollars"* $\rightarrow$ *on may 3 i paid $123*.

The TNP uses the same taxonomy of non-standard words [1] (semiotic classes) as Sproat and Jaitly [2]: *cardinal, decimal, ordinal, measure, money, date, electronic*.

In this paper we focus on showcasing the NTP tool for the task of ITN, but the framework has similar features and support for TN.

The NeMo Text Processing package and English grammars for TN and ITN are open-sourced in the NeMo toolkit [3][1].

## 2. NeMo Text Processing framework

NeMo Text Processing toolkit is a Python extension for *Sparrowhawk* [4][2] - an open-source C++ implementation of the Kestrel TTS text normalization system [5]. Figure 1 gives an overview of the development to deployment process.

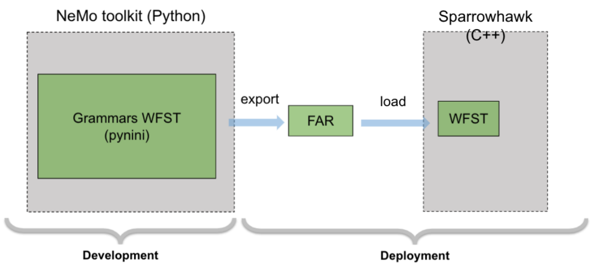


Figure 1: *Schematic diagram of NeMo inverse text normalization development and deployment.*

NTP provides the following features:

- WFST-based grammars. The rule-based system minimizes the number of unrecoverable errors and the input information is preserved. We use *Pynini*, a toolkit built on top of *OpenFst* [6] [3], to define a set of grammars for TN and ITN. More details on the grammar construction could be found in paper [7].
- NTP's design is modular and the grammars are easily extendable to support additional semiotic classes and/or languages.
- English grammars for ITN and TN are included
- NTP provides functionality to export *Pynini* WFST grammars and deploy them with *Sparrowhawk*. The NTP solution was successfully integrated into an NVIDIA ASR product pipeline.

## 3. API design

Following *Sparrowhawk* design we use a two stage normalization pipeline that first detects semiotic tokens (*classification*) and then converts these to written form (*verbalization*).

The framework defines the following APIs that are called in sequence (see Figure 3):

```
classify: str -> str
parse: str -> List[dict]
generate_permutations: List[dict] -> List[str]
verbalize: str -> str
```

*classify()* creates a linear automaton from the input string and composes it with the final classification WFST, which transduces numbers and inserts semantic tags.

*parse()* parses the tagged string into a list of key-value items representing the different semiotic tokens.

*generate_permutations()* is a generator function which takes the parsed tokens and generates string serializations with different permutations of the key-value items. This is important since WFSTs can only process input linearly, but the word order can change from spoken to written form (e.g., *"three dollars"* $\rightarrow$ *$3*).

*verbalize()* takes the intermediate string representation and composes it with the final verbalization WFST, which removes the tags and returns the written form.

Figure 3 depicts the pipeline of ITN conversion.

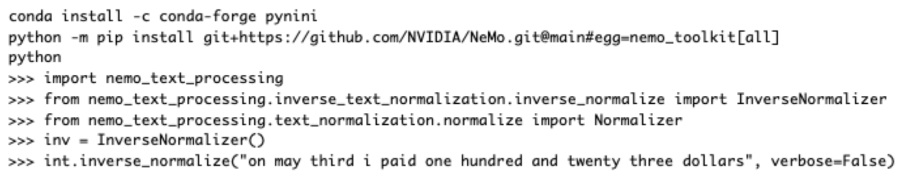
## 4. Usage

### 4.1. Installing Dependencies

NTP relies on NeMo toolkit and *Pynini* library as dependencies. See Figure 2 for installation steps.

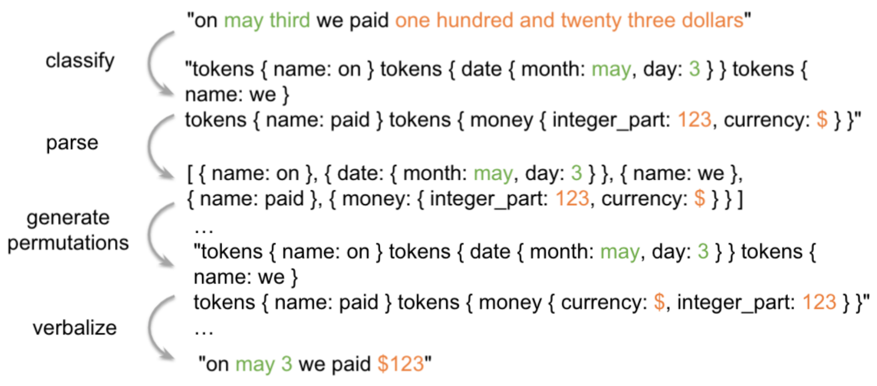
[1]https://github.com/NVIDIA/NeMo/tree/main/nemo_text_processing
[2]https://github.com/google/sparrowhawk

[3]https://github.com/kylebgorman/pynini



```
conda install -c conda-forge pynini
python -m pip install git+https://github.com/NVIDIA/NeMo.git@main#egg=nemo_toolkit[all]
python
>>> import nemo_text_processing
>>> from nemo_text_processing.inverse_text_normalization.inverse_normalize import InverseNormalizer
>>> from nemo_text_processing.text_normalization.normalize import Normalizer
>>> inv = InverseNormalizer()
>>> int.inverse_normalize("on may third i paid one hundred and twenty three dollars", verbose=False)
```

Figure 2: *Dependencies installation steps.*



```
"on may third we paid one hundred and twenty three dollars"

"tokens { name: on } tokens { date { month: may, day: 3 } } tokens {
name: we }
tokens { name: paid } tokens { money { integer_part: 123, currency: $ } }"

[ { name: on }, { date: { month: may, day: 3 } }, { name: we },
{ name: paid }, { money: { integer_part: 123, currency: $ } } ]
…
"tokens { name: on } tokens { date { month: may, day: 3 } } tokens {
name: we }
tokens { name: paid } tokens { money { currency: $, integer_part: 123 } }"
…
"on may 3 we paid $123"
```

Figure 3: *Pipeline of NeMo inverse text normalization.*

### 4.2. Grammar rule extension

We define a Python class for each semiotic class, filled with grammars for that class. Each grammar class is solely responsible for its respective token inputs.

To add a new rule to an existing class, for example to support additional email formats, we can extend $ElectronicFst$ by using *Pynini* primitives. To add an entirely new semiotic class, we create a classification and verbalization grammar class that inherits from the super class $GraphFst$. We append these to the final grammars $ClassifyFst$ and $VerbalizeFst$, respectively.

### 4.3. Deployment to production

We use *Sparrowhawk* as the production backend and demonstrate how to export Pynini grammars and run a provided docker[4] file with Sparrowhawk dependencies installed. The most important file for integration is $semiotic\_classes.proto$ which includes all predefined semiotic classes and their tags. This file needs to be adapted when adding new classes or tags. We provide a script $export\_grammars.sh$ that automatically exports the WFST grammars to the dedicated location and runs *Sparrowhawk* inside a docker container.

## 5. Evaluation

The tool provides capabilities to evaluate the performance of the system using the Google Text Normalization dataset [2]. A large fraction of this dataset can be reused for ITN by swapping input with output but the dataset is not bijective. Therefore, we used regex rules to disambiguate samples where possible. We evaluated the ITN system on the original and disambiguated ('cleaned') dataset using the exact match score, which is the metric most commonly reported for TN [2, 8]. Table 1 summarizes the results.

[4]https://www.docker.com

Table 1: *Results on Google Text Normalization test dataset. We used 92K tokens of the final file.*

| Class | TN | | ITN | |
|---|---|---|---|---|
| | tokens | Accuracy % | tokens | Accuracy % |
| SENTENCE | 6955 | 82.0 | 6955 | 73.7 |
| PLAIN | 62414 | 100.0 | 62284 | 99.1 |
| CARDINAL | 966 | 94.2 | 932 | 99.6 |
| ORDINAL | 95 | 90.5 | 86 | 100.0 |
| DECIMAL | 89 | 100.0 | 89 | 100.0 |
| MEASURE | 127 | 88.2 | 127 | 96.1 |
| MONEY | 29 | 86.2 | 29 | 93.1 |
| DATE | 2607 | 95.9 | 2607 | 94.9 |

## 6. Conclusions

We introduce the NeMo Text Processing toolkit – an open source Python package for inverse text normalization based on WFSTs. NeMo ITN is a Python extension for deployment in *Sparrowhawk*. We provide a set of grammars for English TN and ITN tasks using *Pynini*. The NTP package and the grammars are open-sourced in the NeMo toolkit [3][5].

## 7. Acknowledgements

We would like to thank Richard Sproat for help with Google dataset and *Sparrowhawk* and Kyle Gorman for help with *Pynini*. We would also like to thank our colleagues at NVIDIA for their feedback.

## 8. References

[1] P. Taylor, "Text-to-Speech Synthesis," 2009.

[2] R. Sproat and N. Jaitly, "An RNN model of text normalization," in *INTERSPEECH*, 2017.

[5]https://github.com/NVIDIA/NeMo/tree/main/nemo_text_processing

[3] O. Kuchaiev, J. Li, H. Nguyen, O. Hrinchuk, R. Leary, B. Ginsburg, S. Kriman, S. Beliaev, V. Lavrukhin, J. Cook *et al.*, "Nemo: a toolkit for building ai applications using neural modules," *arXiv:1909.09577*, 2019.

[4] A. Gutkin, L. Ha, M. Jansche, K. Pipatsrisawat, and R. Sproat, "Tts for low resource languages: A bangla synthesizer," in *10th Language Resources and Evaluation Conference*, 2016.

[5] P. Ebden and R. Sproat, "The Kestrel TTS text normalization system," *Natural Language Engineering*, vol. 21, 2015.

[6] K. Gorman, "Pynini: A Python library for weighted finite-state grammar compilation," in *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*, 2016.

[7] Y. Zhang, E. Bakhturina, K. Gorman, and B. Ginsburg, "NeMo Inverse Text Normalization: From Development To Production," *arXiv preprint arXiv:2104.05055*, 2021.

[8] H. Zhang, R. Sproat, A. H. Ng, F. Stahlberg, X. Peng, K. Gorman, and B. Roark, "Neural models of text normalization for speech applications," *Computational Linguistics*, vol. 45, 2019.