PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 109 OCTOBER 2017 5–14

Open-Source Neural Machine Translation API Server

Sander Tars, Kaspar Papli, Dmytro Chasovskyi, Mark Fishel

Institute of Computer Science, University of Tartu, Estonia

Abstract

We introduce an open-source implementation of a machine translation API server. The aim of this software package is to enable anyone to run their own multi-engine translation server with neural machine translation engines, supporting an open API for client applications. Besides the hub with the implementation of the client API and the translation service providers running in the background we also describe an open-source demo web application that uses our software package and implements an online translation tool that supports collecting translation quality comparisons from users.

1. Introduction

The machine translation community boasts numerous open-source implementations of neural (e.g. Junczys-Dowmunt et al., 2016; Sennrich et al., 2017; Helcl and Libovický, 2017; Vaswani et al., 2017), statistical (e.g. Koehn et al., 2007) and rule-based (e.g. Forcada et al., 2011) translation systems. Some of these (e.g. Koehn et al., 2007; Junczys-Dowmunt et al., 2016) even include functionality of server-mode translation, keeping the trained model(s) in memory and responding to the client application's translation requests. However, in most cases the frameworks are tuned for machine translation researchers, and basic production functionality like pre-processing and post-processing pipelines before/after the translation are missing in the translation server implementations.

We present an open-source implementation of a machine translation production server implemented in a modular framework. It supports multiple translation clients running the translation for different language pairs and text domains. The framework consists of:

Corresponding author: fishel@ut.ee

Cite as: Sander Tars, Kaspar Papli, Dmytro Chasovskyi, Mark Fishel. Open-Source Neural Machine Translation API Server. The Prague Bulletin of Mathematical Linguistics No. 109, 2017, pp. 5–14.
doi: 10.1515/pralin-2017-0034.

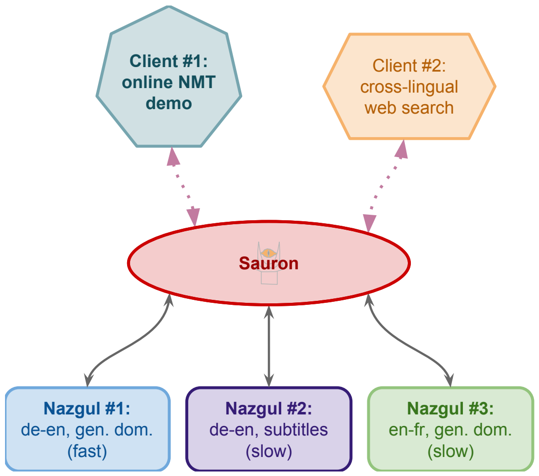

Client #1: online NMT demo
Client #2: cross-lingual web search
Sauron
Nazgul #1: de-en, gen. dom. (fast)
Nazgul #2: de-en, subtitles (slow)
Nazgul #3: en-fr, gen. dom. (slow)

Figure 1. The overall architecture is very simple. Sauron is the server hub, satisfying requests from client applications by querying the translation providers, the Nazgul.

- Sauron: a translation server hub, receiving client requests to translate a text using one of pre-configured translation engines, the Nazgul,
- Nazgul: a translation provider and engine wrapper with custom pre-processing and post-processing steps before/after the translation,
- and a demo web page that uses these two to serve translations to web users, and includes unbiased feedback collection from the users.

The overall architecture is extremely simple and is shown on Figure 1. The hub (Sauron) can serve several clients and is connected to several instances of Nazgul, the translation providers. Each Nazgul is configured to deliver translations for a specific language pair and possibly text domain.

The structure of this paper is the following. Sauron, the translation server hub, is presented in Section 2. Nazgul, the translation engine wrapper is covered in Section 3. The demo web application is described in Section 4. Finally we refer to related work in Section 5 and conclude the paper in Section 6.

2. Sauron, the Translation Server Hub

The central hub tying together all of the components of our framework is Sauron. It works as a reverse proxy, receiving translation requests from client applications and retrieving the translations from one of the Nazgul (which are described in Section 3). The code is freely available on GitHub.[1]

The main features of this central component include

- support for multiple language pairs and text domains
- asynchronous processing of simultaneous translation requests, to enable efficient processing in stressful environments with several requests per second or more
- support for authentication to limit the service only to registered clients if desired
- letting the client application choose between single sentence or whole text translation speed priority

2.1. Client Interface

Access to a running Sauron server is implemented as a simple REST API. Once deployed it runs at a specified URL/IP address and port and supports both GET and POST HTTP communication methods. The API is described and can be tested online on SwaggerHub.[2] The input parameters are:

auth	the authentication token, set in configuration
langpair	a identifier of the source-target language pair, set in configuration
src	the source text
domain	text domain identifier; it can be omitted, leading to the usage of a general-domain translation engine, set in configuration
fast	True indicates the fast, sentence speed-oriented translation method; default is false, document speed-oriented translation
tok	true by default, indicates whether to tokenize the input text
tc	true by default, indicates whether to apply true-casing to the input text
alignweights	false by default, indicates whether to also compute and return the attention weights of the NMT decoder

Although the `fast` parameter is open to interpretation, the idea is to run "fast" translation servers on GPUs, enabling one to focus on the speed of translating a single sentence, while the "slot" servers can be run on CPUs, enabling one to translate a whole document as a batch in multiple threads.

Each combination of language pair, domain and fast/slow has to be covered by a corresponding Nazgul instance, there is no automatic backoff from slow to fast or from in-domain to general domain translation.

[1] `https://github.com/TartuNLP/sauron`

[2] `https://app.swaggerhub.com/apis/kspar/sauron/v1.0`

2.2. Configuration

The only configuration required for Sauron is a list of Nazgul translation provider servers. These are described in an XML file located at `$ROOT/src/main/resources/providers.xml`. Each provider is described with the following parameters:

name	The name, used for system identification in logs
languagePair	A string identifier representing the source-target translation language pair; there is no enforced format but the same string must be used as the value for the API request parameter `lang-pair`
translationDomain	A string identifier representing the translation domain; this is similarly mapped to the API request parameter `domain`
fast	The GPU/CPU preference, a boolean indicating whether the server is using a GPU for translation (whether it is fast); this is mapped to the API request parameter `fast`
ipAddress	The IP address of the translation server
port	The listening port of the translation server

2.3. Deployment

Sauron runs on Java Spring Boot.[3] The preferred method of deployment is to use Gradle[4] to build a `war` file:

```
./gradlew war
```

and deploy it into a Java web container such as Tomcat. You can also run the server without a web container:

```
./gradlew bootRun
```

3. Nazgul, the Translation Servant

Nazgul implements a translation server provider for Sauron. Its design is a modular architecture: every step of the translation service process like pre-processing, translating, post-processing, can be easily modified and substituted. The modularity and open-source format is important for usable machine translation to reduce the

[3]`https://projects.spring.io/spring-boot/`

[4]`https://gradle.org/`

time required to create various application specific services. The code for Nazgul is freely available on GitHub.[5]

Nazgul uses AmuNMT/Marian (Junczys-Dowmunt et al., 2016) as the translation engine (though the modularity of the architecture allows one to replace it easily). The main motivation behind it is because it offers fast neural translation. Moreover, we use a particular modification of this software (available on GitHub[6]), which supports extracting the attention weights after decoding.

3.1. Dependencies

Nazgul is written in Python 2.7 for the reasons of broader compatibility. The implementation requires the following dependencies to be satisfied:

- Downloaded and compiled clone of Marian(AmuNMT) with attention weight output
- The NLTK Python library (Bird et al., 2009). More precisely, the modules `punkt`, `perluniprops` and `nonbreaking_prefixes` are needed. NLTK is used for sentence splitting, tokenization and detokenization[7]

The instructions on how to satisfy these dependencies can be found on the Nazgul GitHub page.[8]

3.2. Deployment

With the dependency requirements satisfied, the server can be run from the command-line simply as a Python file. Example command:

```
python nazgul.py -c config.yml -e truecase.mdl -s 12345
```

The command-line options for running are:

-c	configuration file to be used for AmuNMT run
-e	name of the truecasing model file
-s	the port on which the server will listen (default: 12345)

[5]`https://github.com/TartuNLP/nazgul`

[6]`https://github.com/barvins/amunmt`

[7]To be precise, NLTK uses Moses (Koehn et al., 2007) to tokenize and detokenize by having a Python module nltk.tokenize.moses wrap the Moses tokenizing scripts.

[8]`https://github.com/TartuNLP/nazgul`

The true-caser expects the true-casing models to be trained using the Moses true-caser script.[9] The true-casing model file is expected to be in the same directory with the Nazgul.

The configuration file that is required for AmuNMT translation, is also expected to be in the same directory with the Nazgul. The configuration file specifies the translation model file, vocabularies, whether to use byte pair encoding (BPE, Sennrich et al., 2015), whether to display attention info and many more options. One possible configuration file that we use, is presented on the Nazgul GitHub page with explanations. Additional information can be found on both the original AmuNMT and cloned GitHub pages.

Currently the BPE is only available in Nazgul through AmuNMT configuration file. The reason is that in our experiments having BPE through AmuNMT resulted in faster translation. We are also adding support for separate BPE. To train and apply BPE we used the open-source implementation by Sennrich et al. (2015).[10]

3.3. Workflow

This section describes what happens when Nazgul is started and used to translate. The process is implemented in the file `nazgul.py`.

First, it initialises the key components: AmuNMT, tokenizer, detokenizer, true-caser and finally binds a socket to the specified port to listen for translation requests. Nazgul is capable of serving multiple clients simultaneously.

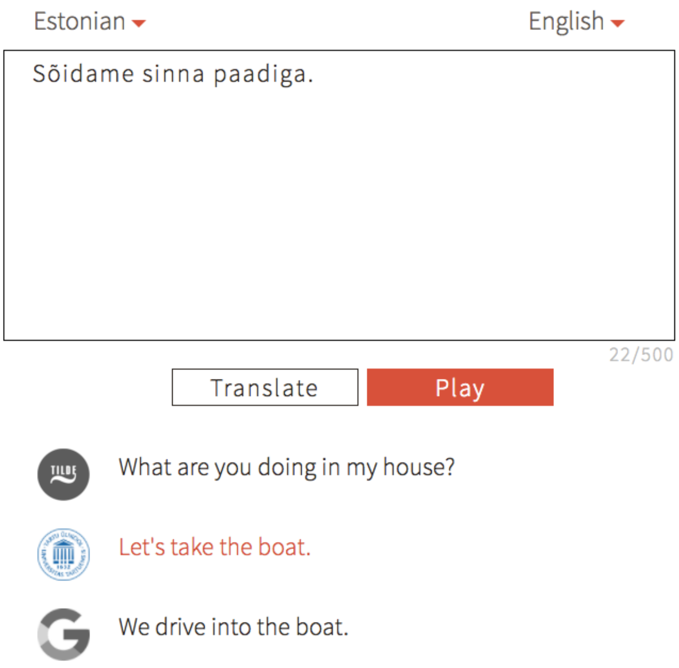
Secondly, when a client connects to Nazgul, the connection is verified and then translation requests are accepted. The necessary protocols are implemented in Sauron, so it is the most convenient option for connecting with Nazgul. For each client connection Nazgul creates a separate thread. The translation request format is a dict in JSON, which includes the fields **src**, **tok** and **tc** that are passed unchanged from Sauron as well as a boolean parameter **alignweights**, which specifies whether this Nazgul should include attention info in the response.

Once the translation request JSON is received, the source string is subjected to pre-processing. Pre-processing starts with sentence splitting, which is always done for the sake of multi-sentence inputs. After that each received sentence is tokenized and truecased, if specified in the JSON input.

After pre-processing, the sentences are sent to the instance of AmuNMT to be translated. From its translation output Nazgul separates the raw translation, attention info, and raw input. It is recommended to disable AmuNMT de-BPE function in the configuration file, otherwise the raw translation will actually be the de-BPEd translation while raw input will be BPEd, thus perturbing the attention info interpretation.

[9] `http://www.statmt.org/moses/?n=Moses.SupportTools#ntoc11`

[10] `https://github.com/rsennrich/subword-nmt`



Sõidame sinna paadiga.

What are you doing in my house?

Let's take the boat.

We drive into the boat.

Figure 2. A screenshot from the web application's Play functionality, which aims to let the users compare the outputs of three translation engines and also to collect the unbiased feedback from the users' selection of the best translation. The Estonian input reads: Let's take the boat there.

When the translation output is received, the translated sentences are subjected to post-processing, which includes detokenization (if tokenization is enabled), and de-truecasing.

Finally, the result of the translation process is sent to the client as a utf-8 encoded JSON dict, which includes fields **raw_trans**, **raw_input**, **weights**, and **final_trans**, which is an array of post-processed and de-BPEd translation outputs. The order of the outputs is the same as in the input text after sentence-splitting.

After sending the response JSON, Nazgul waits for either the next request or termination. Anything that is not JSON is interpreted as a termination signal. In Sauron the process is resolved in such a way that after each fulfilled request the connection is closed. The waiting for next requests is a feature for use cases where the bi-directional communication is expected to have a continuous load for several messages, which would make closing and re-opening the connection an unnecessary overhead.

For further reference on communication, refer to both Nazgul and Sauron documentation pages and simple test scripts presented in the GitHub repository.

4. Neurotõlge, the Example Web Application

Finally we describe an NMT web demo implementation that uses Sauron and Nazgul to fulfill translation requests: Neurotõlge.[11] The demo is live at `http://www.neurotolge.ee` (with an international mirror domain `http://neuralmt.ee`), and the code of the implementation is freely available on GitHub.[12]

The basic functionality of the web application is to translate the input text that the client enters. The text can consist of several sentences, and the client can switch between the available source and target languages (English and Estonian in the live version). Once the client presses the "translate" button the text is translated.

4.1. Collecting User Feedback

Beside the "translate" button there is also a "play" button: once pressed, the application uses three different translation engines to translate the source text. In the live version these are the University of Tartu's translator running on Sauron, Google Translate[13] and Tilde Neural Machine Translation.[14]

Once ready all three translations are displayed in random order without telling the user, which output belongs to which translation engine; the user is invited to select the best translation in order to find out which is which. See an example screenshot of this functionality on Figure 2.

The aim of this feedback collection is to get an unbiased estimation of which translation engine gets selected as best most often. Naturally some users will click on the first or on a random translation, but since the order of the translations is random and the identity of the translation engines is hidden, this will only add uniform noise to the distribution of the best translation engines. This approach was inspired by Blind-Search.[15]

4.2. Dependencies

The front-end of the web application is implemented in JavaScript, using AJAX for asynchronous communications with the back-end and the Bootstrap framework[16] for

[11] *Neural machine translation* in Estonian

[12] `https://github.com/TartuNLP/neurotolge`

[13] `http://translate.google.com/`

[14] `https://translate.tilde.com/neural/`

[15] `http://blindsearch.fejus.com/`

[16] `http://getbootstrap.com/`

an appealing graphic design The back-end is built using Flask.[17] It can be connected to any web server, like Apache, or to be run as a standalone server.

5. Related Work

Some MT service frameworks have been introduced for SMT (Sánchez-Cartagena and Pérez-Ortiz, 2010; Federmann and Eisele, 2010; Tamchyna et al., 2013) and designed to work with Moses (Koehn et al., 2007). The Apertium system also includes a web demo and server framework (Forcada et al., 2011).

NeuralMonkey (Helcl and Libovický, 2017) includes server-running mode, and supports several language pairs and text domains (via different system IDs). However, AmuNMT that our framework uses has been shown to run faster and bringing slightly higher translation quality.

6. Conclusions

We introduce an open-source implementation of a neural machine translation API server. The server consists of a reverse proxy or translation hub that accepts translation requests from client applications and an implementation of a back-end translation server with the pre-processing and post-processing pipelines. The current version uses Marian (AmuNMT) as the translation engine, and the modular architecture of the implementation allows it to be replaced with other NMT engines.

We also described a demo web application that uses the API implementation. In addition to letting its users translate texts it also includes a feedback collection component, which can be used to get an idea of the user feedback on the translation quality.

Future work includes adding a database support to the hub implementation to allow the developer to track the usage of the API, as well as a possibility to visualize the alignment matrix of the NMT decoder on the demo web application to help the users analyze translations and understand, why some translations are counter-intuitive.

Acknowledgements

The projects described here were partially supported by the National Programme for Estonian Language Technology, project *EKT88: KaMa: Kasutatav Eesti Masintõlge / Usable Estonian Machine Translation*.[18]

Bibliography

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

[17] `http://flask.pocoo.org/`

[18] `https://www.keeletehnoloogia.ee/et/ekt-projektid/kama-kasutatav-eesti-masintolge`

Federmann, Christian and Andreas Eisele. MT Server Land: An Open-Source MT Architecure. *The Prague Bulletin of Mathematical Linguistics*, 94:57–66, 2010.

Forcada, Mikel L, Mireia Ginestí-Rosell, Jacob Nordfalk, Jim O'Regan, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Felipe Sánchez-Martínez, Gema Ramírez-Sánchez, and Francis M Tyers. Apertium: a free/open-source platform for rule-based machine translation. *Machine translation*, 25(2):127–144, 2011.

Helcl, Jindřich and Jindřich Libovický. Neural Monkey: An Open-source Tool for Sequence Learning. *The Prague Bulletin of Mathematical Linguistics*, (107):5–17, 2017.

Junczys-Dowmunt, Marcin, Tomasz Dwojak, and Hieu Hoang. Is Neural Machine Translation Ready for Deployment? A Case Study on 30 Translation Directions. *CoRR*, abs/1610.01108, 2016. URL `http://arxiv.org/abs/1610.01108`.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open Source Toolkit for Statistical Machine Translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, 2007.

Sánchez-Cartagena, Víctor and Juan Pérez-Ortiz. ScaleMT: a free/open-source framework for building scalable machine translation web services. *The Prague Bulletin of Mathematical Linguistics*, 93:97–106, 2010.

Sennrich, Rico, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. *CoRR*, abs/1508.07909, 2015. URL `http://arxiv.org/abs/1508.07909`.

Sennrich, Rico, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Valerio Miceli Barone, Jozef Mokry, and Maria Nadejde. Nematus: a Toolkit for Neural Machine Translation. *CoRR*, abs/1703.04357, 2017. URL `http://arxiv.org/abs/1703.04357`.

Tamchyna, Aleš, Ondřej Dušek, Rudolf Rosa, and Pavel Pecina. MTMonkey: A Scalable Infrastructure for a Machine Translation Web Service. *The Prague Bulletin of Mathematical Linguistics*, 100:31–40, 2013.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

Address for correspondence:
Mark Fishel
`fishel@ut.ee`
Institute of Computer Science, University of Tartu
Liivi 2, Tartu 50409
Estonia