**Отчет**

о результатах исследования по использованию

алгоритмов машинного обучения для перевода текста

**Цель исследования:** разработка моделей с программным интерфейсом для машинного перевода исходного текста в целевой с учетом предметной области, определяемой заказчиком продукта.

**Задачи исследования:**

1. Изучить существующие подходы к построению алгоритмов машинного перевода с целью их программной реализации.
2. Провести сравнительный анализ существующих моделей машинного перевода для выбора наиболее применимых в разработке программного интерфейса.
3. Сформировать перечень источников наборов данных для обучения моделей машинного перевода.
4. Определить аппаратное обеспечение для применения алгоритмов машинного обучения моделей при обработке естественного языка.
5. Разработать управляемый механизм разработки моделей машинного перевода, который позволит в целевом языке учитывать предметную область заказчика.
6. Разработка программного интерфейса для машинного перевода исходного текста в целевой

# 1. Существующие подходы к построению алгоритмов машинного перевода

Существует несколько принципиально разных подходов к построению алгоритмов машинного перевода: основанный на правилах (rule-based), статистический, или основанный на статистике (statistical-based), нейронный машинный перевод (neural machine translation, NMT). Последний подход получил наибольшее развитие благодаря развитию современных архитектур нейронных сетей и распространению решений с открытым исходным кодом.

В открытом доступе существует значительное число ресурсов с готовыми решениями нейронного машинного перевода. Наиболее популярными решениями являются:

|  |  |  |  |
| --- | --- | --- | --- |
| **Решение** | **Язык** | **Фреймворк** | **Число языков** |
| Tensor2Tensor | Python | TensorFlow |  |
| FairSeq | Python | PyTorch |  |
| Nmt | Python | TensorFlow |  |
| OpenNMT | Python/C++ | PyTorch/TensorFlow |  |
| Sockeye | Python | MXNet |  |
| Nematus | Python | Tensorflow |  |
| Marian | C++ | – | 177 |
| THUMT | Python | PyTorch/TensorFlow |  |
| NMT-Keras | Python | Keras |  |
| Neural Monkey | Python | TensorFlow |  |
| Transformers Hub | Python | PyTorch/TensorFlow/JAX | 177 |

В ходе проведенного исследования были проведены эксперименты с Marian для машинного перевода.

Marian написан на чистом C ++ [1]. Основной зависимостью Marian является Boost. Marian может быть скомпилирован на машинах с устройствами NVIDIA GPU и CUDA 8.0+ или на машинах с CPU. Версия Marian для ЦП компилируется автоматически, если обнаружены OpenBLAS или Intel MKL (рекомендуется).

Эксперименты показали, что полноценное использование моделей для Marian можно свободно реализовать под OS Linux. Однако в OS Windows могут возникнуть проблемы, связанные с выполнением дополнительных скриптов по предобработке исходного текста; без такой предобработки может быть получен перевод, который меняет смысл исходного текста (например, «документом о Кремлях» вместо корректной фразы «документом о взглядах Кремля»; «встретить репрессивную политику государства» вместо корректной фразы «противостоять репрессивной политике государства»).

Кроме этого, при использовании Marian требуется выполнять компиляцию под целевую операционную систему, что снижает гибкость его использования, в т.ч. при обучении моделей. Так, для одной эпохи при обучении на центральном процессоре (Intel(R) Core(TM) i9-9900 CPU @ 3.10GHz, 3096 МГц, ядер: 8, логических процессоров: 16) с корпусом из 4468840 парных предложений ушло 27 часов 44 минуты.

Дальнейшие исследования определили поиск более гибких решений, среди которых следует выделить [Transformers](https://pypi.org/project/transformers/), которое предоставляет доступ к набору данных, словарям и предварительно обученным моделям машинного перевода (размещены на сайте [huggingface.co](https://huggingface.co/)), их дообучения и локального использования. Экосистема Transformers включает в себя набор инструментов для работы с набором данных, распределенного использования аппаратного обеспечения для машинного обучения и оценки моделей (Transformers, Datasets, Tokenizers и Accelerate).

Сравнительный анализ Marian c Transformers в скорости перевода [тестового набора](https://github.com/eleldar/Translator/raw/master/test_dataset/test_dataset.xlsx) показал следующие результаты (Intel(R) Core(TM) i9-9900 CPU @ 3.10GHz, 3096 МГц, ядер: 8, логических процессоров: 16):

|  |  |  |
| --- | --- | --- |
|  | **Англо-русский** | **Арабско-русский** |
| Marian | 99,1 сек. | 94,6 сек. |
| Transformers | 183,4 сек. | 152,7 |
| Разница (абс/отн) | 84,3 / 1,8 | 58.1 / 1,6 |

Из таблицы видно, что Marian работает значительно быстрее. Однако отмеченные выше ограничения на использование его моделей для обучения и практического применения определил Transformers в качестве более перспективной основы для решения дальнейших задач исследования.

# 2. Сравнительный анализ существующих моделей машинного перевода

Существующие модели машинного перевода можно разделить по следующим критериям:

* архитектура модели;
* гиперпараметры, использованные при обучении;
* набор данных, использованный при обучении;
* метод, использованный при формировании словаря (токенизация);
* число языков, которые может обработать модель.

Основная часть современных моделей основана на трансформерах [2, 3, 4, 5]. Другие модели, как правило, используются в исследованиях, чтобы их сравнить, в том числе с трансформерами. В научных работах можно встретить тестирование таких моделей как amun и s2s [6, 7].

Основные архитектуры современных моделей машинного перевода:

|  |  |  |  |
| --- | --- | --- | --- |
| **Модель** | **Кодер** | **Декодер** | **Сложность при обучении** |
| [RCTM 1 (Kalchbrenner and Blunsom, 2013)](https://www.sciencedirect.com/science/article/pii/S2666651020300024#bib64) | CNN | RNN | O(S2+T) |
| [RCTM 2 (Kalchbrenner and Blunsom, 2013)](https://www.sciencedirect.com/science/article/pii/S2666651020300024" \l "bib64) | CNN | RNN | O(S2+T) |
| RNNEncdec/Seq2Seq (Cho et al., 2014a; Sutskever et al., 2014) | RNN | RNN | O(S+T) |
| [RNNsearch (Bahdanau et al., 2015)](https://www.sciencedirect.com/science/article/pii/S2666651020300024#bib10) | RNN | RNN | O(ST) |
| [ByteNet (Kalchbrenner et al., 2016)](https://www.sciencedirect.com/science/article/pii/S2666651020300024" \l "bib65) | CNN | CNN | O(S+T) |
| [ConvSeq2Seq (Gehring et al., 2017)](https://www.sciencedirect.com/science/article/pii/S2666651020300024" \l "bib43) | CNN | CNN | O(ST) |
| [Transformer (Vaswani et al., 2017)](https://www.sciencedirect.com/science/article/pii/S2666651020300024" \l "bib126) | SAN | SAN | O(S2+ST+T2) |

где RNN - recurrent neural network, CNN - convolution neural network, SAN - and self-attention network, S – длина слов исходного языка, T - длина слов целевого языка

На сайте [huggingface.co](https://huggingface.co/) размещено 1444 моделей машинного перевода с архитектурой Transformer. В разрезе моделей наибольшее число принадлежит следующим:

|  |  |
| --- | --- |
| **Модель** | **Количество** |
| marian | 1367 |
| T5 | 20 |
| mbart | 11 |

Исходя из этого в качестве предобученных моделей были взяты модели marian. Исследования показали, что среди этих моделей сейчас доступно 1339 языковых пар, из которых:

* уникальные языковые пары – 1330, повторы – 9.
* симметричные пары – 469 (всего 938 моделей).
* ассиметричные – 194.
* мультиязычные – 198.

Полный список доступных моделей сведен в [таблицу](https://github.com/eleldar/Translator/raw/master/table.xlsx). При этом их количество и содержание не статично, поскольку могут добавляться новые либо обновляться существующие.

# 3. Набор данных для обучения моделей машинного перевода

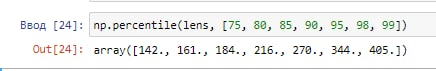
Из открытого доступа были загружены следующие корпуса:

- [англо-русского языков](https://object.pouta.csc.fi/Tatoeba-Challenge-v2021-08-07/eng-rus.tar) (217 612 504 предложений);

- [арабско-русского языков](https://object.pouta.csc.fi/Tatoeba-Challenge-v2021-08-07/ara-rus.tar) (60 875 366 предложений);

- [арабско-английского языков](https://object.pouta.csc.fi/Tatoeba-Challenge-v2021-08-07/ara-eng.tar) (126 590 524 предложений).

Для работы было сформировано 3 файла в формате CSV с длиной 405, не превышающие 405 символа. Данная длина выбрана на основе ограничений, которые вводятся на длину входных данных при обучении и описательной статистики для данных корпусов:



Для применения сформированных наборов данных в обучении моделей были созданы объекты с использованием модуля [memmap](https://numpy.org/doc/stable/reference/generated/numpy.memmap.html) библиотеки NumPy. Однако итоговый объем данных составил более 900 Гб, что вызвано особенностями организации хранения данных этой библиотекой.

С целью применения более эффективных решений по организации хранения и использования набора данных в настоящее время проводится изучение двух основных подходов: Apache Arrow (рис. 1) и генераторы потока (рис. 2).

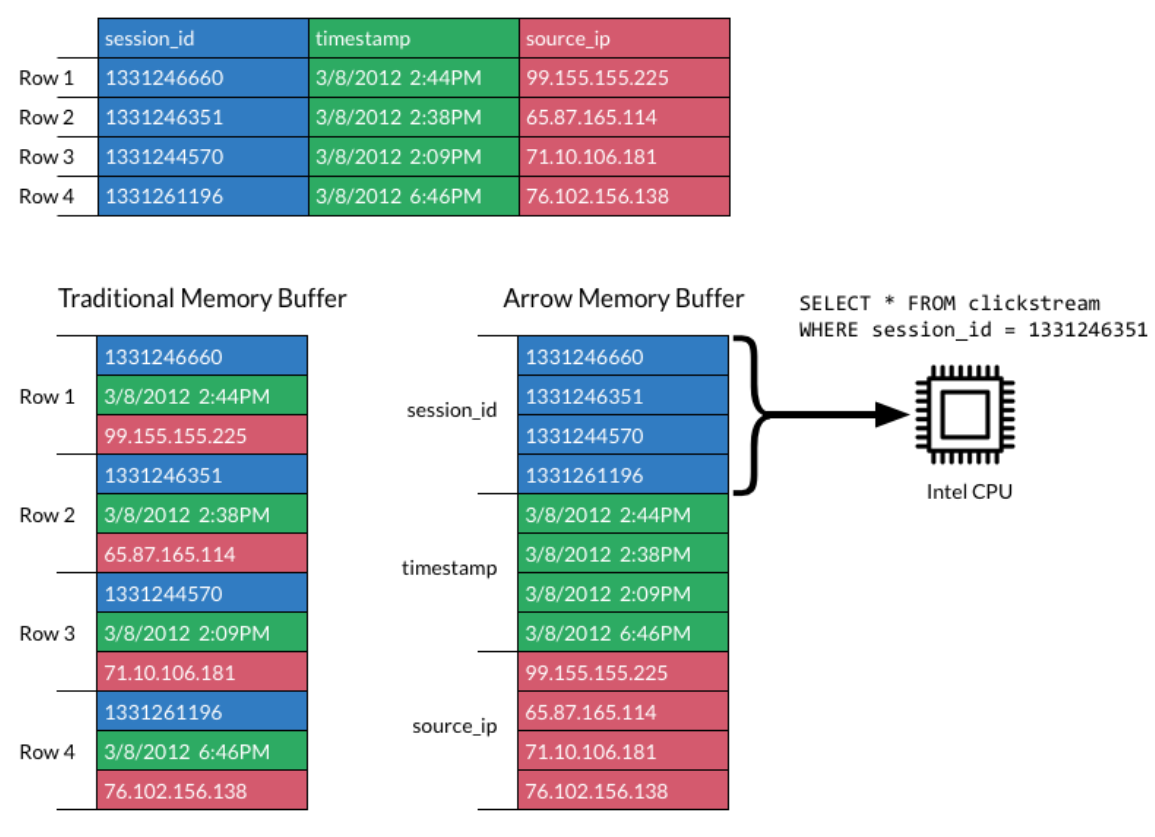


Рис. 1. Сравнение Apache Arrow с традиционным подходом к хранению данных

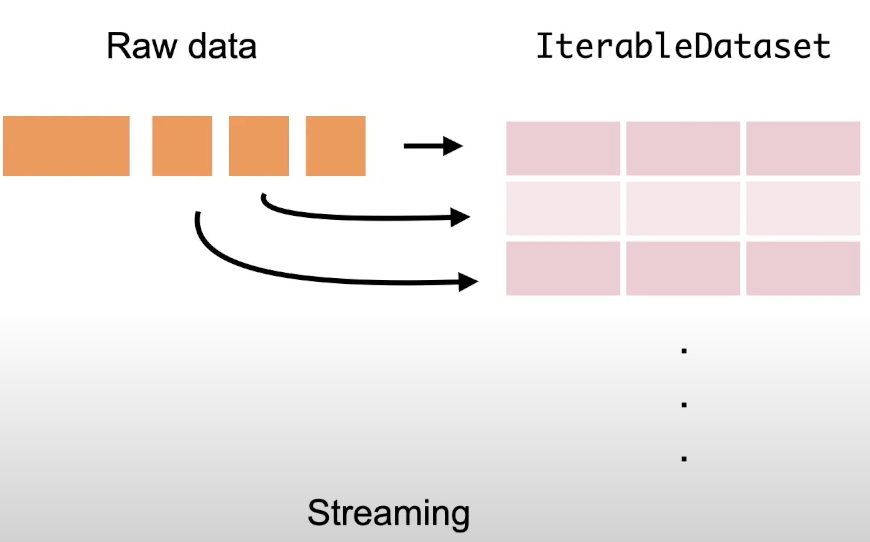


Рис. 2. Схема создания генератора потока набора данных

Оба подхода не требуют создания новых файлов, однако имеют свои преимущества и недостатки. Так, для создания объекта в оперативной памяти из корпуса 217 610 088 англо-русских предложений Apache Arrow потребовал 181 Мб, для создания генератора потока потребовалось всего 9 Мб. Вместе с тем, скорость обращения к элементам первого объекта в разы быстрее, чем ко второму.

С генератором потока набора данных был проведен эксперимент по подсчёту числа элементов, состоящий из 210 607 728 текстов (в архиве занимает 425 ГБ). Обычные циклы не справились с этой задачей в течение 24 часов, поэтому было применено распределенное вычисление на 16 логических процессорах, которое заняло 7 часов 15 минут.

# 4. Аппаратное обеспечение для применения алгоритмов машинного обучения моделей при обработке естественного языка

Для выбора оптимального аппаратного обеспечения для обучения моделей машинного обучения на учебных данных был проведен эксперимент в Google Colab Pro, который показал следующие результаты. Для обработки корпуса из 1377 парных предложений потребовалось:

- CPU - 9 часов 15 минут;

- GPU (Tesla P100-PCIE) - 10 минут 16 секунд;

- TPU - 1 час 22 минуты.

Выбор Google Colab обусловлен тем, что для обучения в Kaggle не хватает памяти при выполнении алгоритма по обучению модели:



Аналогичная проблема возникает при попытке применить библиотеку для распределения ресурсов:



Аналогичный эксперимент проведен на локальном сервере с одним GPU (GeForce RTX 208). Скорость работы по сравнению c Google Colab Pro составила 7 минут 7 секунд.

Также следует отметить, что ключевой особенностью алгоритмов по обработке естественных языков является то, что они могут передавать различные размеры пакетов в модель. Эта особенность не позволяет реализовать в полной мере TPU для обучения моделей машинного обучения.

Исходя из этого, оптимальным выбором аппаратного обеспечения для обучения моделей машинного перевода является GPU.

# 5. Управляемый механизм разработки моделей машинного перевода

Создание управляемого механизм разработки моделей машинного перевода можно разбить на три основных направления:

1. Разработка абсолютно новых словаря и токенизатора с последующим обучением модели, веса которой были инициализированы случайными значениями;
2. Использование существующего токенизатора с последующим обучением модели, веса которой были инициализированы случайными значениями;
3. Использование существующего токенизатора с последующим обучением предобученной модели.

В рамках последнего направления был проведен эксперимент и в существующей модели удалось добиться желаемого результата.

Для обучения использовалось 79916 парных предложений, которые применены к модели 'Helsinki-NLP/opus-mt-en-ru'. Обучение проходило в течение 2-х эпох на GPU (Tesla P100-PCIE) на протяжении 2:37 часов.

Полученные результаты:

|  |  |  |
| --- | --- | --- |
| **Исходный текст** | **До обучения** | **После обучения** |
| Security of online payments | Обеспечение онлайновых платежей | Безопасность онлайн-платежей |

При сравнении словарей токенизаторов в marian на С++ с marian на Python выяснилось, что они одинаковые (отличие лишь в служебном символе Transformers). Поэтому для дообучения на этом токенизаторе придется выполнять предобработку либо строить свой токенизатор (но в этом случае дообучение будет равнозначно обучению с нуля, т.к. IDs вероятнее всего будут другими).

# 6. Программный интерфейс для машинного перевода

Для реализации программного интерфейса был разработан RESTful API на основе библиотек Python (flask и flask\_restful) с документированием спецификации по стандарту OpenAPI. В настоящее время документированный сервис доступен по адресу: <http://10.7.102.48:8888/api/>. Запросы прослушиваются по адресу: <http://10.7.102.48:8888/api/api/translation>.

Для работы с интерфейсом требуется передать 3 аргумента: исходный текст, код исходного языка и код целевого языка (например: "text": "Hello World!", "sourceLanguage": "en", "targetLanguage": "ru"). В ответ API возвращает этот запрос с ключом "message", содержащий переведенный текст.

Настоящее время доступны следующие направления перевода: en-ru, ru-en, ar-ru, ru-ar, en-ar, ar-en, где "en" – Английский, "ru" – Русский, "ar" – Арабский.

С текущей реализацией сервиса был проведен эксперимент для перевода английского и арабского текста на русский из [тестового набора](https://github.com/eleldar/Translator/raw/master/test_dataset/test_dataset.xlsx).

|  |  |  |
| --- | --- | --- |
| **Англо-русский перевод** | **Всего** | **Среднее** |
| Предобработка | 1,08 мс | 0,016 мс |
| Токенизация | 21,63 мс | 0,318 мс |
| Вычисление вектора | 34424,31 мс | 506,24 мс |
| Декодирование | 29,69 мс | 0,437 мс |
| Использование RAM | - | 13,87 Мб |

|  |  |  |
| --- | --- | --- |
| **Арабско-русский перевод** | **Всего** | **Среднее** |
| Предобработка | 1,04 мс | 0,018 мс |
| Токенизация | 19,51 мс | 0,342 мс |
| Вычисление вектора | 33238,76 мс | 583,136 мс |
| Декодирование | 27,17 мс | 0,477 мс |
| Использование RAM | - | 18,15 Мб |

Сравнить с работой Marian точно не получается, поскольку требуется сборка с реализацией функций по измерению каждого этапа. Произвольные замеры потребляемой памяти с момента запуска цикла перевода в Linux показали, что она колеблется от 20 до 60 Мб.

# Список использованных источников

1. [Marian: Fast Neural Machine Translation in C++ / Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak et al. // Proceedings of ACL 2018, System Demonstrations. — Melbourne, Australia: Association for Computational Linguistics, 2018. — July. — Pp. 116–121](https://aclanthology.org/P18-4020.pdf). http://www.aclweb.org/anthology/P18-4020.

2. [From Research to Production and Back: Ludicrously Fast Neural Machine Translation / Young Jin Kim, Marcin Junczys-Dowmunt, Hany Hassan et al. — 2019.](https://github.com/eleldar/Translator/blob/master/articles/d19_5632.pdf)

3. [Molchanov, Alexander. PROMT Systems for WMT 2018 Shared Translation Task / Alexander Molchanov. — 2019](https://github.com/eleldar/Translator/blob/master/articles/w18_6420.pdf).

4. [The AFRL IWSLT 2020 Systems: Work-From-Home Edition / Brian Ore, Eric Hansen, Tim Anderson, Jeremy Gwinnup. — 2020](https://github.com/eleldar/Translator/blob/master/articles/2020iwslt_111.pdf).

5. [Translating Between Morphologically Rich Languages: An Arabicto-Turkish Machine Translation System / ˙Ilknur Durgar El-Kahlout, Emre Bekta¸s, Naime ¸Seyma Erdem, Hamza Kaya. — 2019](https://github.com/eleldar/Translator/blob/master/articles/w19_4617.pdf).

6. [Marian: Cost-effective High-Quality Neural Machine Translation in C++ / Marcin Junczys-Dowmunt, Kenneth Heafield, Hieu Hoang et al. — 2019](https://github.com/eleldar/Translator/blob/master/articles/w18_2716.pdf).

7. [Tambouratzis, George. Alignment verification to improve NMT translation towards highly inflectional languages with limited resources / George Tambouratzis, Marina Vassiliou. — 2021](https://github.com/eleldar/Translator/blob/master/articles/2021eacl_main158.pdf).