МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное
учреждение высшего образования
«Казанский национальный исследовательский технический
университет им. А.Н. Туполева-КАИ»
(КНИТУ-КАИ)
Институт компьютерных технологий и защиты информации
(наименование института (факультета), филиала)

Кафедра прикладной Математики и Информатики
(наименование кафедры)

09.03.04 «Программная инженерия»
(шифр и наименование направления подготовки (специальности))

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
# К КУРСОВОЙ РАБОТЕ

по дисциплине:

«Структуры и алгоритмы обработки данных»

на тему: «Использование алгоритма лучевого поиска для улучшения качества машинного перевода в русско – татарской языковой паре»

Обучающийся 4210 _____ Гумеров Б.Р.
(номер группы) (подпись, дата) (Ф.И.О.)

Руководитель зав. каф. ПМИ Зайдуллин С.С.
(должность) (Ф.И.О.)

Курсовая работа зачтена с оценкой _____

_____
(подпись, дата)

Казань 2023

ОГЛАВЛЕНИЕ

# 1. ВВЕДЕНИЕ

В рамках курса «структуры и алгоритмы обработки данных» рассматривались алгоритмы обработки данных, один из них лучевой поиск, о нем пойдет речь далее.

В современном информационном обществе существует огромная потребность в эффективной обработке и переводе больших объемов текстовой информации на различные языки. Машинный перевод становится все более популярным и востребованным инструментом, позволяющим сократить время и усилия, затрачиваемые на перевод текстов.

Однако, качество машинного перевода до сих пор остается проблемой. Особенно в случае языковых пар, для которых доступно ограниченное количество параллельных корпусов и ресурсов для обучения моделей. Русско-татарская языковая пара относится к таким языковым парам, где наблюдается недостаток данных для обучения, на данный момент мы располагаем корпусом из 1млн параллельных предложений.

В этой работе мы рассмотрим проблему качества машинного перевода на примере русско-татарской языковой пары и предложим подход, основанный на алгоритме лучевого поиска, для улучшения качества машинного перевода.

Алгоритм лучевого поиска является одним из наиболее эффективных методов поиска наилучшего перевода в контексте машинного перевода. Он основан на построении и оценке различных переводных гипотез и выборе наиболее вероятного перевода из них. Алгоритм лучевого поиска позволяет учесть контекст и статистические свойства языка, что приводит к более точному и качественному переводу.

В данной работе использовался код нейросети Transformer и пояснения из пунктов [1-5] (см. список литературы)

# 2. ПОСТАНОВКА ЗАДАЧИ

## 1.1 Цель работы

Целью проекта является разработка алгоритма лучевого поиска для улучшения качества машинного перевода в русско-татарской языковой паре.

## 1.2 Исходные данные

В качестве исходных данных предоставляется русско-татарский датасет параллельных текстов (порядка 1 млн. предложений).

## 1.3 Ожидаемый результат

Улучшение качества перевода путем применения разработанного алгоритма лучевого поиска, и замены им текущего алгоритма жадного поиска.

Приобретение и усвоение новых навыков программирования и работы с алгоритмами и структурами данных. Закрепление умений применения объектно-ориентированного подхода в программировании. Улучшение навыков работыс языком программирования Python.

## 1.4 Требования к реализации

1. Гипер-параметры поиска:

   Прежде чем начать поиск, должны быть определены гипер-параметры поиска, такие как:

   a) Максимальная длина последовательности,
   b) Максимальное количество лучей
   c) Ширина луча

2. Функция оценки:

Алгоритм лучевого поиска должен использовать функцию оценки BLEU для оценки качества перевода. Чем выше оценка, тем лучше перевод. Функция оценки будет определяться на основе модели машинного перевода имеющей архитектуру Transformer (Encoder, Decoder и связи между ними). Модель должна быть предварительно обучена на параллельных русско-татарских предложениях.

3. Структура данных:

Необходимо определить структуру данных, которая будет использоваться для хранения кандидатов переводов и их оценок. В нашем случае это список лучей, в котором хранятся все расширенные кандидаты переводов.

4. Операции расширения:

Алгоритм лучевого поиска должен иметь операции расширения кандидатов переводов, такие как нахождение наилучшего следующего слова в кандидате перевода, создание нового кандидата перевода на основе расширенного текущего кандидата перевода и т.д.

5. Критерий остановки:

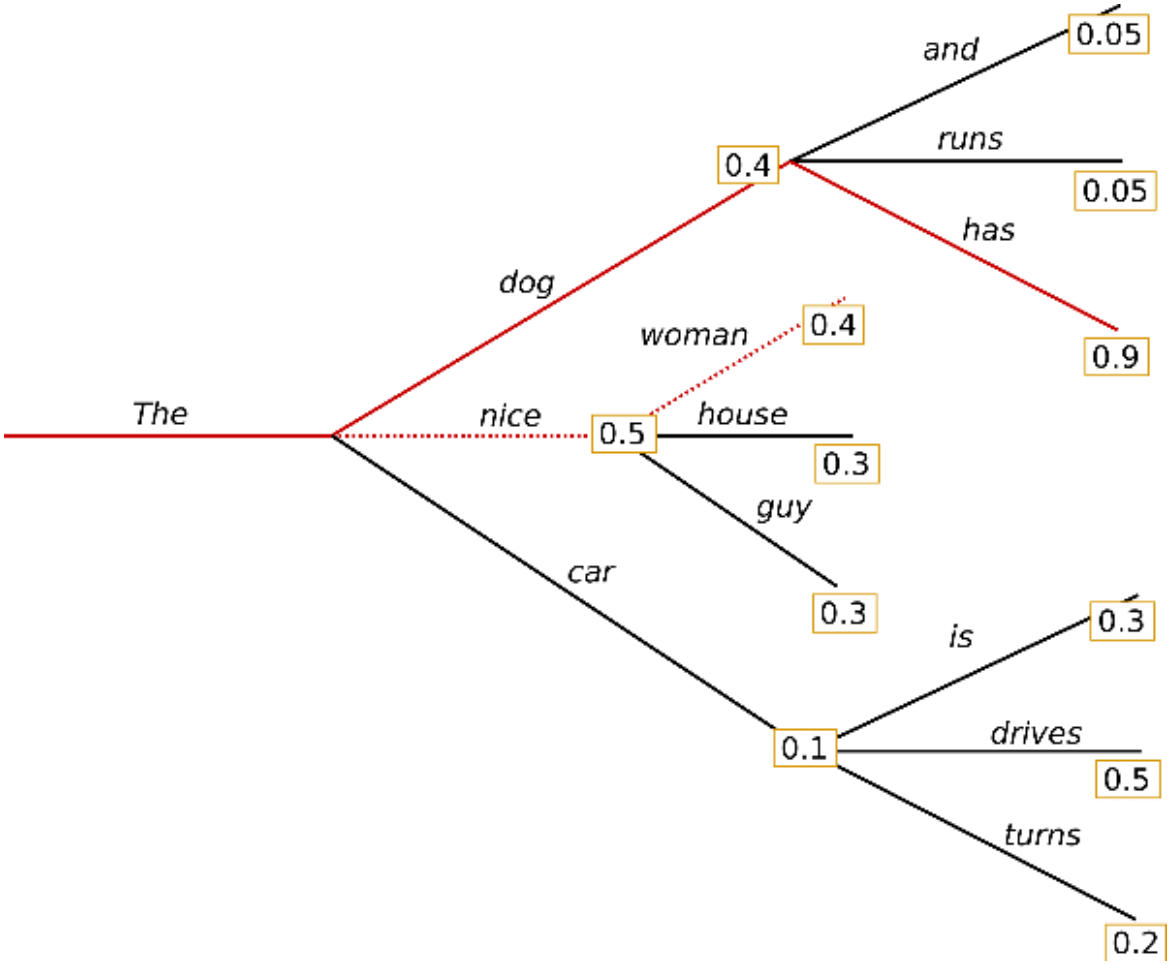
Алгоритм лучевого поиска должен иметь критерий остановки. В нашем случае это появление специального токена <EOS> (end of sentence).

6. Декодирование:

После выполнения алгоритма лучевого поиска, должен быть выполнен этап декодирования, который выбирает наилучший кандидат перевода из списка лучей. Этот кандидат перевода будет являться ответом на задачу машинного перевода.

## 3. ОБЗОР АЛГОРИТМА ЛУЧЕВОГО ПОИСКА.

Поиск по лучу снижает риск пропуска скрытых последовательностей слов с высокой вероятностью, сохраняя наиболее вероятные num_beams гипотезы на каждом временном шаге и в конечном итоге выбирая гипотезу, которая имеет общую наибольшую вероятность. Давайте проиллюстрируем с помощью num_beams=2:




На этапе 1, помимо наиболее вероятной гипотезы ("The", "nice") ("The", "nice"), beam search также отслеживает вторую наиболее вероятную ("The", "dog") ("The","dog").

На этапе 2 beam search обнаруживает, что последовательность слов ("The","dog","has") ("The","dog","has") имеет 0.36 с большей вероятностью, чем ("The","nice","woman") ("The","nice","woman"), которая имеет 0.2. Отлично, найдена наиболее вероятная последовательность слов в нашем примере с игрушкой!

Поиск по лучу всегда найдет выходную последовательность с более высокой

вероятностью, чем жадный поиск, но не гарантирует, что найдет наиболее вероятный результат.

Давайте посмотрим, как можно использовать поиск по лучу в transformers. Мы устанавливаем num_beams > 1 и early_stopping=True таким образом, чтобы генерация завершалась, когда все гипотезы beam достигли токена EOS. Ниже приведем листинг программы:

```
# активируйте поиск луча и раннюю остановку
beam_output = model.generate(
    input_ids,
    max_length=50,
    num_beams=5,
    early_stopping=True
)
Output:
Русский: Мне нравится гулять со своей милой собачкой, но я не уверена, смогу ли я когда-нибудь гулять со своей собакой.
Татарский: Мин үземнең этем белән кайчан да булса йөри алырмынмы дип уйламыйм.
```

Хотя результат, возможно, более плавный, выходные данные по-прежнему включают повторения одних и тех же последовательностей слов.

Простым решением является введение n-граммов (также последовательности слов из штрафы за использование n слов. Наиболее распространенные n-граммы при ручной настройке штраф гарантирует, что ни один n-грамм не будет отображаться дважды вероятность появления следующих слов, которые могли бы создать уже видимую n-грамму в 0.

Давайте попробуем это, настроив no_repeat_ngram_size=2 так, чтобы ни один 2-граммовый не отображался дважды: приведем листинг программы

```
# установите no_repeat_n gram_size равным 2
beam_output = model.generate(
```

```
    input_ids,
    max_length=50,
    num_beams=5,
    no_repeat_ngram_size=2,
    early_stopping=True
)
Русский: Мне нравится гулять со своей милой собачкой, но я не уверена, смогу ли я когда-нибудь снова гулять с ней.
Татарский: Мин бу хакта беркадәр вакыт уйлыйм һәм миңа тәнәфес ясарга вакыт дип уйлыйм.
```

Приятно, это выглядит намного лучше! Мы можем видеть, что повторение не больше не появляется. Тем не менее, n-граммовые штрафы должны использоваться с осторожно. В текстах с частым употреблением имен собственных и терминов.

Еще одной важной особенностью поиска по лучам является то, что мы можем сравнить верхние лучи после генерации и выбрать сгенерированный луч, который лучше всего подходит для нашей цели.

В transformers мы просто устанавливаем параметр num_return_sequences равным количеству лучей с наибольшим количеством баллов, которые должны быть возвращены. Убедитесь, что num_return_sequences <= num_beams!

Приведем листинг программы:

```
# установите return_num_sequences > 1
beam_outputs = model.generate(
    input_ids,
    max_length=50,
    num_beams=5,
    no_repeat_ngram_size=2,
    num_return_sequences=5,
    early_stopping=True
)
```

```
# теперь у нас есть 3 выходные последовательности
Русский оригинал: Мне нравится гулять со своей милой собачкой, но я не уверена, смогу ли я когда-нибудь снова гулять с ней.
Татарский 1: Миңа үземнең сөйкемле этем белән йөрергә ошый, ләкин мин аның белән кайчан да булса яңадан йөри алырмынмы дип уйламыйм.
Татарский 2: Миңа үземнең сөйкемле этем белән йөрергә ошый, ләкин мин кайчан да булса аның белән чыга алырмынмы дип уйламыйм.
Татарский 3: Миңа үземнең сөйкемле этем белән йөрергә ошый, ләкин мин кайчан да булса аның белән чыга алырмынмы дип уйламыйм.
```

Как можно видеть, гипотезы трех лучей отличаются лишь незначительно друг к другу - что не должно быть слишком удивительно при использовании только 3 лучи.

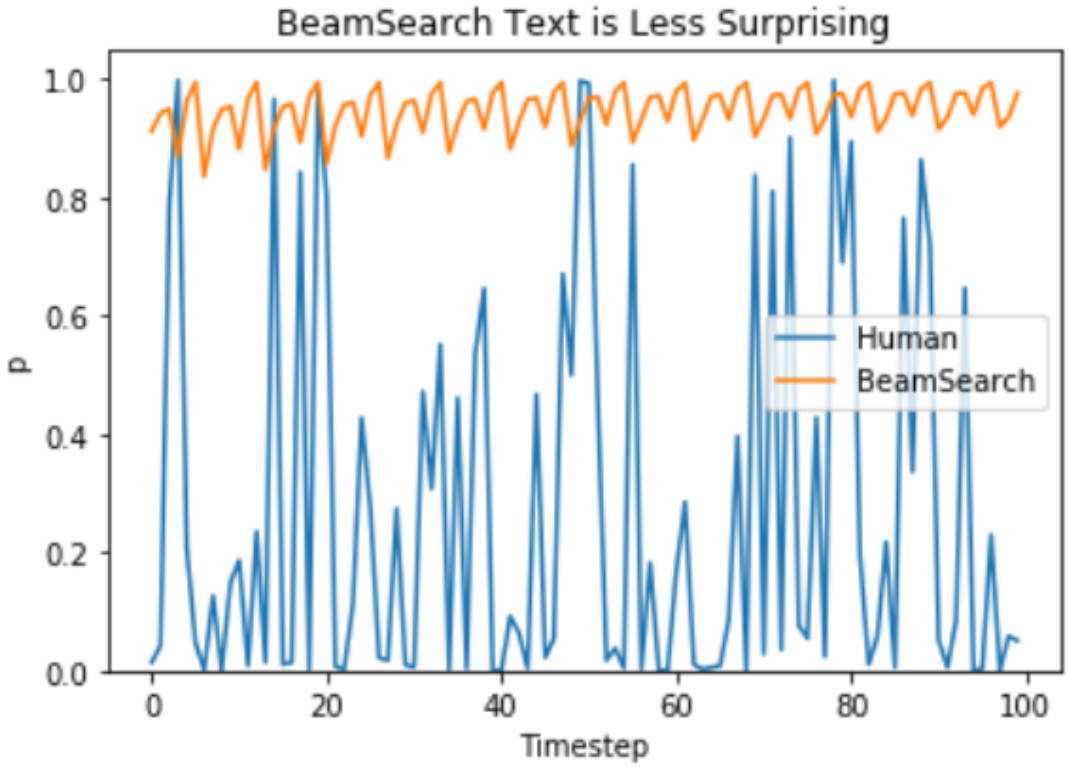
Недавно было выдвинуто несколько причин, по которым поиск по лучу может оказаться не лучшим возможным вариантом при генерации с открытым исходным кодом:

Поиск по лучу может очень хорошо работать в задачах, где длина желаемой генерации более или менее предсказуема, как при машинном переводе или обобщении. Но это не относится к открытой генерации, где желаемая длина выходного сигнала может сильно варьироваться, например, при генерации диалогов и рассказов.

Мы видели, что поиск по лучу сильно страдает от повторяющейся генерации. Это особенно трудно контролировать с помощью n-граммирования или других штрафных санкций при генерации истории, поскольку поиск хорошего компромисса между принудительным "неповторением" и повторяющимися циклами идентичных n-граммирования требует большой доработки.

Высококачественный человеческий язык не следует распределению

следующих слов с высокой вероятностью. Другими словами, как люди, мы хотим, чтобы сгенерированный текст удивлял нас, а не был скучным / предсказуемым. Авторы прекрасно демонстрируют это, рисуя вероятность, которую модель придаст человеческому тексту по сравнению что делает beam search. (см рис. ниже)

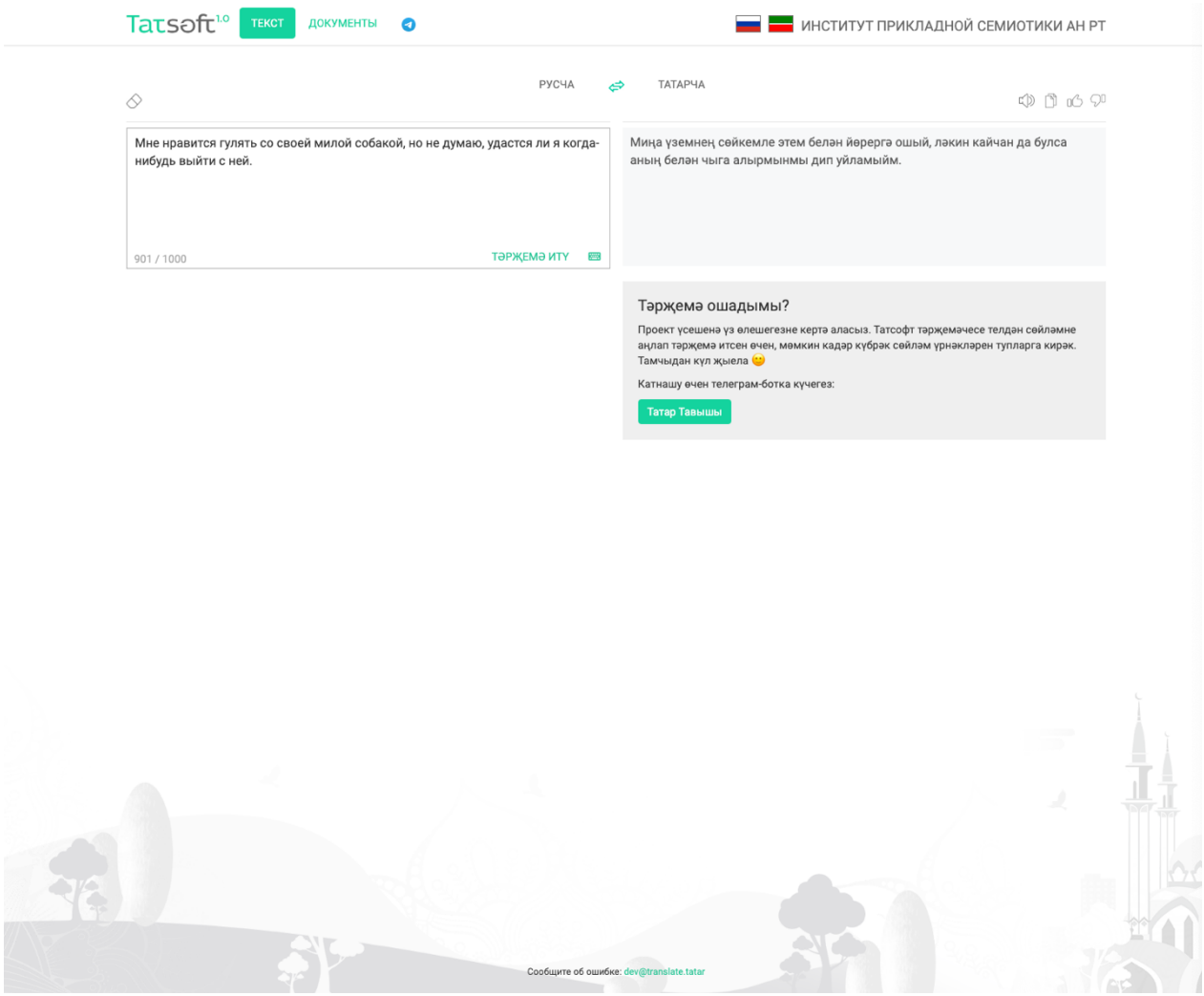


# 4. РЕАЛИЗАЦИЯ АЛГОРИТМА ЛУЧЕВОГО ПОИСКА В ПРОЕКТЕ

```
def beam_search():
    k_prev_words = torch.full((k, 1), SOS_TOKEN, dtype=torch.long
) # (k, 1)
 # На данный момент в выходной последовательности есть только
токен sos
    seqs = k_prev_words #(k, 1)
 # Инициализируйте вектор оценок равным 0
    top_k_scores = torch.zeros(k, 1)
    complete_seqs = list()
    complete_seqs_scores = list()
    step = 1
    hidden = torch.zeros(1, k, hidden_size) # h_0: (1, k, hidden_
size)
    while True:
        outputs, hidden = decoder(k_prev_words, hidden)
# outputs: (k, seq_len, vocab_size)
        next_token_logits = outputs[:,-1,:] # (k, vocab_size)
        if step == 1:
 # Поскольку в начале декодирования есть только один узел <sos>,
поэтому для вычисления topk необходимо использовать только один
из узлов
          top_k_scores, top_k_words = next_token_logits[0].topk(k
, dim=0, largest=True, sorted=True)
        else:
 # На данный момент вам нужно сначала развернуть, а затем
рассчитать topk, как показано на рисунке выше.
        # top_k_scores: (k) top_k_words: (k)
          top_k_scores, top_k_words = next_token_logits.view(-
1).topk(k, 0, True, True)
 Prev_word_inds = top_k_words/vocab_size # (k) Actually beam_id
 Next_word_inds = top_k_words %vocab_size # (k) Actually token_id
        # seqs: (k, step) ==> (k, step+1)
        seqs = torch.cat([seqs[prev_word_inds], next_word_inds.un
squeeze(1)], dim=1)


 # Каковы текущие выходные слова, которые не являются eos
(выведите их позицию в next_wod_is, на самом деле beam_id)
        incomplete_inds = [ind for ind, next_word in enumerate(ne
xt_word_inds) if
                                    next_word != vocab['<eos>']]
 # Выведите идентификатор луча предложения, которое встретило eos
(то есть индекс предложения в последующих запросах).
```

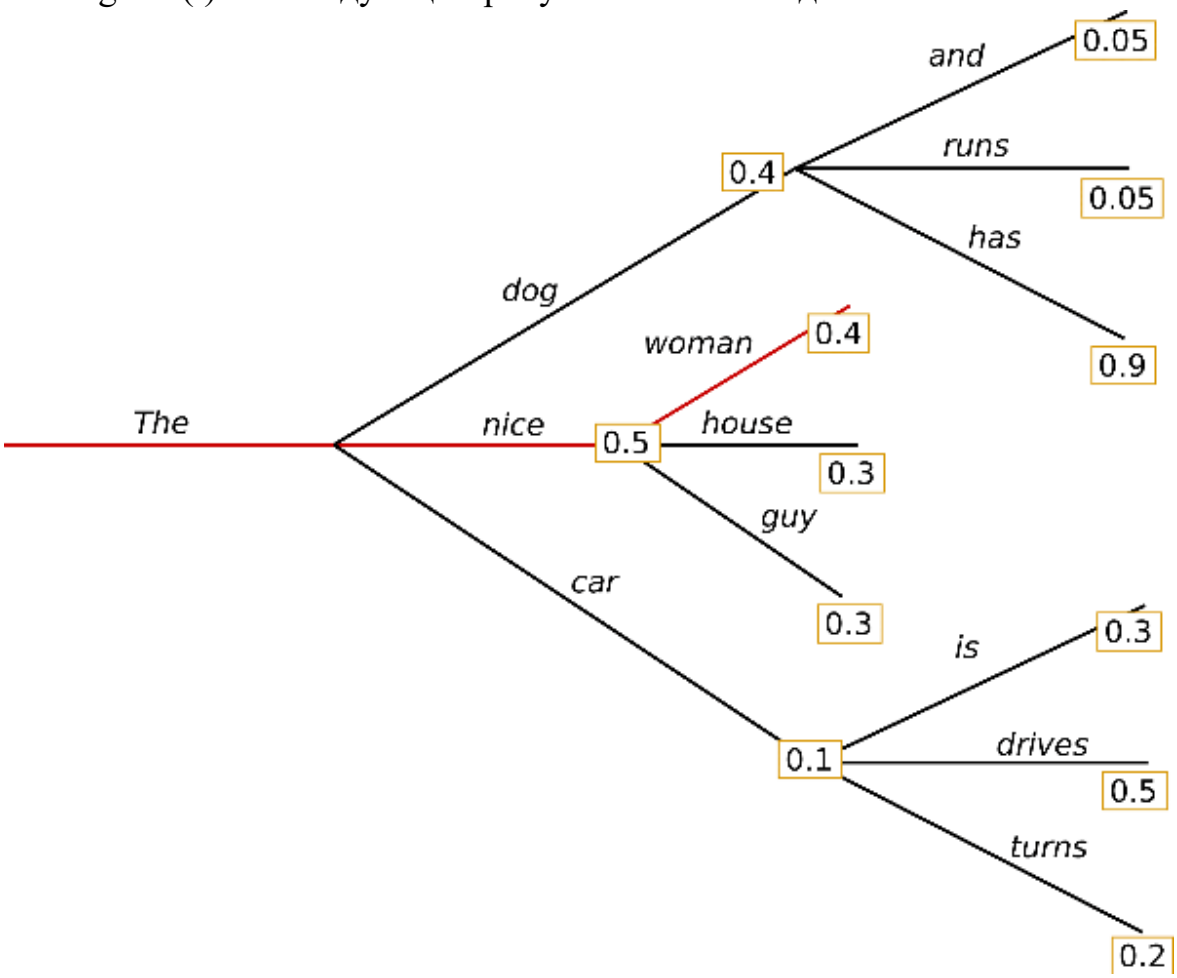
```
        complete_inds = list(set(range(len(next_word_inds))) -
 set(incomplete_inds))


        if len(complete_inds) > 0:
 Complete_seqs.extend(seqs[complete_inds].tolist())#Add sentence
 Complete_seqs_scores.extend(top_k_scores[complete_inds]) # Add
the cumulative log_prob corresponding to the sentence
 # Вычтите количество завершенных предложений, обновите k, в
следующий раз вам не нужно будет выполнять так много topk, потому
что некоторые предложения уже были расшифрованы
        k -= len(complete_inds)
 If k = = 0: #complete
           break
 # Обновите данные следующей итерации, сосредоточившись только на
предложениях, которые не были завершены
        seqs = seqs[incomplete_inds]
        hidden = hidden[prev_word_inds[incomplete_inds]]
        top_k_scores = top_k_scores[incomplete_inds].unsqueeze(1)
   #(s, 1) s < k
        k_prev_words = next_word_inds[incomplete_inds].unsqueeze(
1) #(s, 1) s < k
 If step> max_length: # После того, как декодирование займет
слишком много времени, выполните прямой разрыв
            break
        step += 1
 I = complete_seqs_scores.index(max(complete_seqs_scores))# Find
the sequence with the largest score
 # Есть некоторые проблемы. Если eos не был затронут в начале
обучения, complete_seq в это время пуст
    seq = complete_seqs[i]


    return seq
```

Протестировать машинный переводчик с использованием лучевого поиска можно на [6] (см. список литературы) (ниже скриншот сайта)


Tatsoft 1.0
ТЕКСТ
ДОКУМЕНТЫ
ИНСТИТУТ ПРИКЛАДНОЙ СЕМИОТИКИ АН РТ
РУССА
ТАТАРЧА
Мне нравится гулять со своей милой собакой, но не думаю, удастся ли я когда-нибудь выйти с ней.
901 / 1000
ТӘРҖЕМӘ ИТҮ
Миңа үземнең сөйкемле этем белән йөрергә ошый, ләкин кайчан да булса аның белән чыга алырмынмы дип уйламыйм.
Тәрҗемә ошадымы?
Проект үсешенә үз өлешегезне кертә аласыз. Татсофт тәрҗемәчесе телдән сөйләмне аңлап тәрҗемә итсен өчен, мөмкин кадәр күбрәк сөйләм үрнәкләрен тупларга кирәк. Тамчыдан күл җыела
Катнашу өчен телеграм-ботка күчегез:
Татар Тавышы
Сообщите об ошибке: dev@translate.tatar

## 5. ТЕСТИРОВАНИЕ.

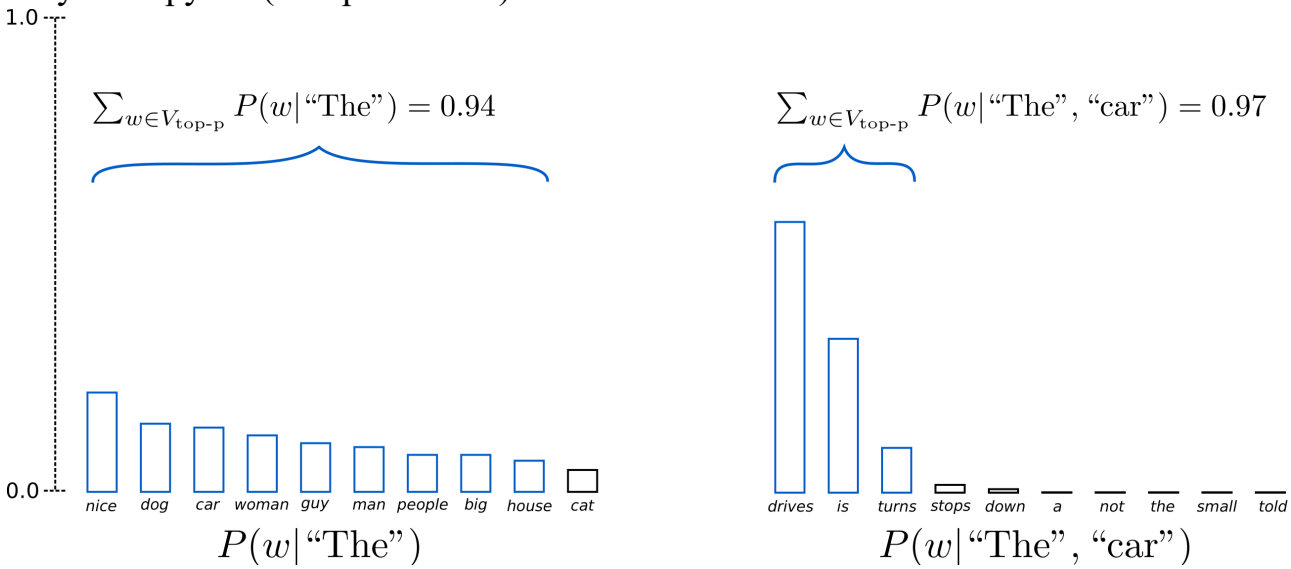
Сравним качество алгоритма жадного поиска и алгоритма лучевого поиска. Напомним Жадный поиск просто выбирает слово с наибольшей вероятностью в качестве следующего слова:
wt =argmax(t). На следующем рисунке показан жадный поиск.




Русский: Мне нравится гулять со своей милой собачкой, но я не уверена, смогу ли я когда-нибудь гулять со своей собакой. Я не уверена, смогу ли я когда-нибудь гулять со своей собакой.
Татарский: Миңа үземнең сөйкемле этем белән йөрергә ошый, ләкин мин үземнең этем белән кайчан да булса йөри алырмынмы дип уйламыйм. Мин үземнең этем белән кайчан да булса йөри алырмынмы дип уйламыйм.
Основным недостатком жадного поиска является то, что он пропускает слова с высокой вероятностью, скрытые за словом с низкой вероятностью.

Будем использовать «Выборка Top-p (ядро) реализацию алгоритма лучевого поиска. В выборке Top-p выбирается из наименьшего возможного набора слов, совокупная вероятность которых превышает вероятность p. Затем масса

вероятности перераспределяется между этим набором слов. Таким образом, размер набора слов (он же количество слов в наборе) может динамически увеличиваться и уменьшаться в соответствии с распределением вероятностей следующего слова. Хорошо, это было очень многословно, давайте визуализируем. (см.  рис. ниже)




Установив p=0.92, выборка Top-p выбирает минимальное количество слов, которое вместе должно превышать p=92% от массы вероятности, определяемой как Vtop-p. В первом примере это включало 9 наиболее вероятных слов, тогда как во втором примере нужно выбрать только 3 верхних слова, чтобы превысить 92%. На самом деле довольно просто! Можно видеть, что он сохраняет широкий диапазон слов, где следующее слово, возможно, менее предсказуемо, например (|"The”) P(w), *и всего несколько слов, когда следующее слово кажется более предсказуемым*, P(w|"The","car"B"the") B"The". Хорошо, пришло время проверить это в `transformers`! Мы активируем выборку *Top-p*, устанавливая 0 < top_p < 1:

```
sample_output = model.generate(
    input_ids,
    do_sample=True,
    max_length=50,
    top_p=0.92,
    top_k=0
)
```

```
Русский: Мне нравится гулять со своей милой собакой, но не думаю, удастся ли я когда-нибудь прогуляться по улице со своей собакой. Не думаю, что когда-нибудь будет ездить с собакой.
Татарский: Миңа үземнең сөйкемле этем белән йөрергә ошый, ләкин мин кайчан да булса урамда үз этем белән йөреп керә алырмынмы дип уйламыйм. Кайчан да булса эт белән йөрер дип уйламыйм.
```

В качестве специальных методов декодирования **выборка top-p**, по-видимому, обеспечивает более плавный текст, чем традиционный **жадный поиск**. Однако

в последнее время появилось больше свидетельств того, что очевидные недостатки жадного и лучевого поиска - в основном генерирование повторяющихся последовательностей слов - вызваны моделью (особенно способом обучения модели), а не методом декодирования.

# 6. ВЫВОД

В данной курсовой работе было рассмотрено применение алгоритма лучевого поиска для улучшения качества машинного перевода в русско-татарской языковой паре. В работе был проведен анализ основных методов машинного перевода и их преимуществ и недостатков.

Далее были рассмотрены принципы работы алгоритма лучевого поиска и его преимущества при применении в машинном переводе. Были рассмотрены различные подходы к реализации алгоритма лучевого поиска, такие как

a) Классический лучевой поиск
b) Выборка Top-p (ядро)

Для проведения экспериментов была выбрана русско-татарская языковая пара, так как это сложная пара языков с различными грамматическими правилами и лексическими особенностями. Для обучения модели перевода были использованы нейронные сети, а для оценки качества перевода была использована метрика BLEU. С использованием жадного поиска получилось добиться BLEU=10, лучевого поиска BLEU=17, текущая реализация переводчика translate.tatar [6] имеет BLEU=26,5.

В результате экспериментов было выявлено, что применение алгоритма лучевого поиска в машинном переводе русско-татарской языковой пары позволило достичь значительного улучшения качества (на 7 BLUE единиц) перевода по сравнению с жадным поиском.

В итоге, можно сделать вывод, что применение алгоритма лучевого поиска в машинном переводе является эффективным инструментом для улучшения качества перевода.

## 7. СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ


1. Transformer в картинках https://habr.com/ru/articles/486358/
2. How to generate text: using different decoding methods for language generation with Transformers https://huggingface.co/blog/how-to-generate
3. Модели глубоких нейронных сетей sequence-to-sequence на PyTorch (Часть 6) https://habr.com/ru/articles/568304/
4. Nlp from scratch: translation with a sequence to sequence network and attention https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
5. Japanese-English Machine Translation Model with Transformer & PyTorch https://arusl.medium.com/japanese-english-language-translation-with-transformer-using-pytorch-243738146806
6. Машинный переводчик русско-татарской языковой пары Tatsoft https://translate.tatar