Записки научных
семинаров ПОМИ
Том 552, 2026 г.

Г. Э. Ангени, В. А. Ершов

КЛИЕНТООБУСЛОВЛЕННОЕ СПЕКУЛЯТИВНОЕ ДЕКОДИРОВАНИЕ ДЛЯ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ В МУЛЬТИАРЕНДНЫХ СЕРВИСАХ

§1. Введение

Большие языковые модели (large language models, LLM) играют ключевую роль в работе огромного числа сервисов, и с течением времени количество продуктов, использующих большие языковые модели, только растет. Однако многие компании, разрабатывающие сервисы на основе больших языковых моделей, не способны выделить ресурсы на обучение и поддержку собственных LLM, и по этой причине они обращаются к облачным сервисам – третьим сторонам, предоставляющим доступ к необходимым моделям в режиме, оговорённом в SLA (service-level agreement). Как правило, в SLA входят гарантии по таким техническим показателям, как задержка до первого токена и среднее время на токен. Для поддержания своей конкурентноспособности, облачный сервис стремится, при условии соблюдения SLA со всеми своими клиентами, предоставлять свои услуги по наиболее выгодным ценам, в связи с чем, в частности, данные сервисы являются мультиарендными, то есть обслуживают множество клиентов, обрабатывая их запросы с помощью одних и тех же мощностей, не создавая отдельные экземпляры под клиента.

При условии отсутствия SLA одним из очевидных способов минимизации издержек на работу сервиса (с последующими снижениями цен за услуги) является уменьшение объёма вычислительных мощностей в облаке и увеличение числа запросов, отправляемых на каждую машину. В таком случае генерация ответов будет замедляться пропорционально объёму уменьшения мощностей, однако для того, чтобы не прибегать к поддержке чрезмерно большого объёма оборудования с соблюдением SLA, приходится прибегать к разнообразным техникам ускорения работы самих больших языковых моделей, позволяющим

Ключевые слова: спекулятивное декодирование, ReDrafter, мультиарендные сервисы, персонализация, пропускная способность, acceptance rate, n-граммные модели.

добиться “удешевления” их работы с минимальными изменениями в качестве генерации. Несмотря на то, что данные техники действительно применяются в промышленных сценариях, в первую очередь они направлены на интегральное ускорение модели, то есть оптимизацию её применения вне зависимости от пришедшего запроса без учёта его содержания, но в мультиарендных сценариях содержание потока запросов от каждого клиента может сыграть существенную роль, поскольку, несмотря на то, что совокупный поток запросов обладает сложным распределением, отдельные клиенты пользуются облачными сервисами в рамках узких сценариев.

В рамках данной работы рассматривается подход к большим языковым моделям именно в контексте их применения в составе мультиарендного сервиса. В качестве базового подхода к увеличению пропускной способности модели, не оказывающего влияния на качество ответов, выбрано спекулятивное декодирование. Проведено исследование современных методов спекулятивного декодирования, а также предложена гипотеза об увеличении пропускной способности системы с большой языковой моделью путём встраивания информации о клиенте. Предложены методы, реализующие данную гипотезу при помощи обусловливания распределения токенов, возвращаемых при спекулятивном декодировании, за счёт введения априорного распределения, уникального для каждого клиента, вычисляемого по n-граммным статистикам, собираемым с ответов модели на его запросы в онлайн-режиме. Выбран подход к спекулятивному декодированию ReDrafter, на базе исходного кода которого были внедрены предложенные методы[1]. Проведен ряд экспериментов, в ходе которых установлено улучшение как теоретической метрики acceptance rate, так и целевой метрики пропускной способности модели.

§2. Обзор области

2.1. Спекулятивное декодирование. В первой работе про спекулятивное декодирование [4] исследуется идея использования легковесной, но быстрой, драфтовой модели (draft model) в совокупности с тяжеловесной, но более качественной, целевой моделью (target model)

[1]Исходный код с реализованными методами доступен в репозитории `https://github.com/geaned/ml-recurrent-drafter`

для совместной генерации текста. Суть совместной генерации заключается в том, что кандидат на продолжение текущей последовательности, генерируемый драфтером, состоит из нескольких токенов, а целевая модель верифицирует всё продолжение за одно применение, после чего с помощью спекулятивного сэмплирования принимается некоторый префикс кандидата. Алгоритм выглядит следующим образом:

(1) Выполнить prefill-стадию на первоначальной последовательности токенов $t_1, ..., t_n$, просэмплировать токен t_{n+1}, а также сохранить скрытые состояния целевой модели;
(2) Использовать драфтовую модель и скрытые состояния целевой модели для генерации продолжения $t'_{n+2}, ..., t'_{n+\gamma+1}$ с соответствующими распределениями $q_{n+2}, ..., q_{n+\gamma+1}$, из которых данные токены были получены;
(3) Применить целевую модель ко всей последовательности $t_{n+1}, ..., t_{n+\gamma+1}$ с целью получения распределений $p_{n+2}, ..., p_{n+\gamma+2}$;
(4) Для каждой пары (p_i, q_i), начиная с $i = n+2$ и заканчивая $i = n+\gamma+1$, выполнить алгоритм спекулятивного сэмплирования и в результате либо принять токен t_i и перейти к следующей паре распределений (при наличии), либо отвернуть токен t'_i и просэмплировать токен на этой позиции заново (и "отбросить" все распределения, начиная с индекса $i+1$);
(5) Продолжить первоначальную последовательность $t_1, .., t_{n+1}$ токенами $t'_{n+2}, ..., t'_{n+l+1}$, где l – длина принятого префикса, просэмплировать токен t_{n+l+2} (в лучшем случае будет просэмплирован токен $t_{n+\gamma+2}$ из распределения $p_{n+\gamma+2}$) и перейти к шагу 2 в случае, если end-of-sentence токен ещё не был сгенерирован.

Нетрудно видеть, что при идеальной генерации кандидатов драфтовой моделью, скорость генерации текста такой системой увеличивается в $\frac{\gamma+1}{\gamma c+1}$ раз относительно использования целевой модели, при этом качество генерации остается неизменным (здесь и далее γ – длина продолжения-кандидата в токенах, c – отношение временных затрат на одно применение драфтовой модели ко времени на применение целевой модели), однако если драфтовая модель никогда не угадывает токены целевой модели, то скорость изменяется в $\frac{1}{\gamma c+1} < 1$ раза. Стоит отметить, что для того, чтобы гарантировать полное сохранение генераций такой системой относительно целевой модели, достаточно

Сэмплирование с отклонением

- Вычислить $M = \max\limits_{x \in V} \frac{p(x)}{q(x)}$, где V – словарь токенов;
- Сэмплировать $x \sim q(x)$ и $r \sim U[0, 1]$;
- Если $r < \frac{p(x)}{Mq(x)}$, то принять x, иначе повторить с предыдущего шага.

Спекулятивное сэмплирование

- Сэмплировать $x \sim q(x)$;
- Если $q(x) \leqslant p(x)$, то принять x, иначе сэмплировать $r \sim U[0, 1]$;
- Если $r < \frac{p(x)}{q(x)}$, то принять, иначе сэмплировать x из нормализованного распределения:

$$\frac{\max(0, p(x) - q(x))}{\sum\limits_{x \in V} \max(0, p(x) - q(x))}$$

Таблица 1. Сравнение алгоритмов сэмплирования.

осуществить одну итерацию сэмплирования с отклонением (rejection sampling) и сэмплировать токен напрямую из распределения целевой модели в случае неудачи, однако использование спекулятивного сэмплирования более оптимально с точки зрения доли принимаемых токенов (или же acceptance rate, α). В таблице 1 приведены оба алгоритма сэмплирования.

Для сэмплирования с отклонением вероятность принятия токена можно посчитать напрямую: $\alpha_{rej} = \mathbb{E}_{x \sim q(x)} = \mathbb{E}_x p(x) \min x' \frac{q(x')}{p(x')}$, при этом авторы статьи выводят, что при использовании спекулятивного сэмплирования вероятность принятия равна $\alpha_{spec} = \sum\limits_x \min(p(x), q(x))$, и в общем случае значение α_{spec} существенно выше α_{rej}.

Несмотря на абсолютно новую идею для оптимизации больших языковых моделей и множество теоретических выкладок, в рамках данной работы не было проведено значительное число экспериментов – в качестве целевой модели была взята T5-XXL [8], а в качестве драфтовой модели было проверено несколько вариантов из того же семейства.

Прирост скорости генерации был экспериментально продемонстрирован на датасетах, также использованных в статье про семейство моделей T5, в задачах перевода и суммаризации текстов на датасетах, соответственно, EnDe и CNN/DM. Наибольший прирост, а именно 3.4 раза при сэмплировании с нулевой температурой и 3.1 раза при единичной температуре, был получен при использовании самой легковесной модели из семейства – T5-Small.

Стоит отметить, что при использовании данного метода также увеличивается время, необходимое для генерации первого токена. Данная метрика особенно важна в сценариях взаимодействия с конечным пользователем, поскольку с увеличением задержки реакции системы на его запрос увеличивается риск нарушения SLA, вследствие чего клиент может перестать пользоваться предоставляемой системой. По этой причине помимо максимизации α стоит научиться минимизировать c, например, путём использования модифицированных архитектур, в которых метод спекулятивного декодирования внедряется напрямую в целевую модель (аналогично, например, адаптерам [3]) и переиспользует её состояние. Далее в рамках обзора речь пойдет о подобных модификациях.

2.2. Medusa. Данный метод опирается на соображение о том, что при применении большой языковой модели скрытые состояния после последних блоков декодера хорошо вбирают в себя общий смысл поданного на вход текста, и поэтому можно генерировать драфт, основываясь на h_n – последнем скрытом состоянии модели, за котором в архитектуре больших языковых моделей следует “голова” (language-modeling head), отображающая его в логиты (logits) распределения токенов. На основании последних скрытых состояний авторы обучают серию из γ легковесных моделей, каждая из которых представляет собой несколько идущих друг за другом полносвязных слоев, после которых идёт голова. Таким образом, i-я драфтовая модель независимо от других моделирует распределение q_{n+i+1}, что позволяет применять головы параллельно. Это наглядно продемонстрировано на рисунке 1. Также подобная архитектура драфтовой модели позволяет осуществлять генерацию кандидатов с минимальной дополнительной нагрузкой, в отличие от применения отдельного трансформера под эту задачу.

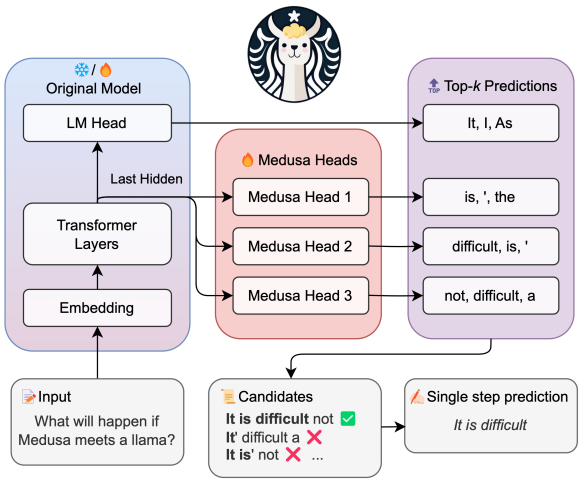

Рис. 1. Процесс применения Medusa. i-я голова предсказывает наиболее вероятные токены на $n + i + 1$-ой позиции, $n + 1$-ый токен предсказан целевой моделью (n – длина входной последовательности в токенах). Элементы списка кандидатов являются представителями декартового произведения списков наиболее вероятных токенов на каждой позиции, однако все кандидаты начинаются с самого вероятного $n+1$-го токена по распределению целевой модели.

В качестве способа увеличения доли принятых токенов α авторы предлагают проверять большой языковой моделью не только последовательность, состоящую из самых вероятных токенов (на каждой позиции – по мнению соответствующей драфтовой модели), а любую из последовательностей, где на $n + i + 1$-ой позиции стоит токен из s_i наиболее вероятных токенов по распределению, полученному из соответствующей головы. Иными словами, рассматривается декартово произведение $T_1 \times ... \times T_\gamma$ размера $s_1 \cdot ... \cdot s_\gamma$, где T_i соответствует множеству из наиболее вероятных токенов по результатам предсказания i-ого драфтера. Для того, чтобы в рамках одного батча все последовательности рассматривались корректно, в целевую модель внедряется

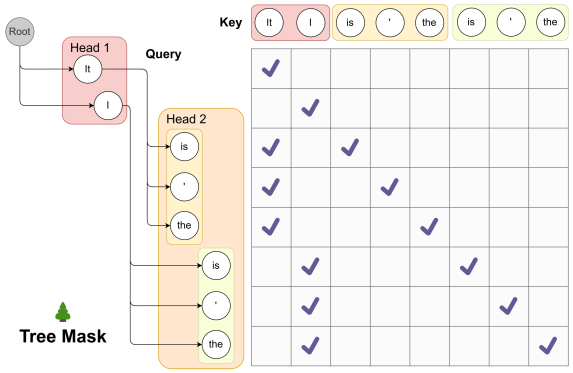

Рис. 2. Наглядное представление маски tree attention для $\gamma = 2, s_1 = 2, s_2 = 3$. Все кандидаты складываются в батч и верифицируются целевой моделью одновременно. В каждой строке ненулевой коэффициент внимания возможен только для токенов, принадлежащих определенному продолжению или его префиксу (отмечены галочками).

древовидная маска внимания (tree attention), позволяющая целевой модели обработать всех кандидатов независимо друг от друга. Модифицированная маска внимания изображена на рисунке 2.

Авторы указывают на то, что им удалось добиться увеличения числа выходных токенов в секунду в 2.18 раза для модели Vicuna-7B и в 2.33 раза для Vicuna-13B. Помимо этого, при дообучении целевой модели вместе с Medusa прирост составляет 2.83 раза для обеих моделей, однако при использовании такого подхода, безусловно, меняется качество генерации, поэтому на его реализации внимание в рамках обзора заостряться не будет. Важно отметить, что данная техника нацелена исключительно на улучшение системы с точки зрения скорости генерации, поскольку при увеличении γ, а также значений s_i, размер батча будет увеличиваться экспоненциально. В свою очередь, это делает данную технику неприменимой в сценариях с параллельной обработкой множества потоков запросов даже при помощи видеокарт с большим количеством видеопамяти.

Для того чтобы минимизировать издержки с сохранением пользы, получаемой от использования tree attention, авторы предлагают рассматривать не всё декартово произведение множеств наиболее вероятных токенов, а жадным образом выбрать подмножество кандидатов, имеющих наибольший шанс быть принятыми целевой моделью. Для этого модель с обученными драфтерами запускается на отдельном валидационном датасете и считаются доли принятия $a_k^{(i)}$, где k – номер головы ($1 \leqslant k \leqslant \gamma$), а i – порядковый номер по убыванию вероятности токена ($1 \leqslant i \leqslant s_i$). Далее строится дерево верифицируемых последовательностей по следующему алгоритму:

(1) Начальное дерево представляет собой одну вершину, соответствующую токену t_{n+i+1}, который был предсказан целевой моделью и относительно которого строится продолжение; фиксируется значение S – искомый размер дерева без учёта этой вершины;
(2) В дерево добавляется вершина, которая наилучшим образом увеличивает оценку длины принятой последовательности, вычисляемую по формуле:

$$\sum_{l=1}^{k} \sum_{[i_1,\ldots,i_l]\in I} \prod_{j=1}^{l} a_j^{i_j}$$

Фактически выражение представляет собой сумму вероятностей всех последовательностей, входящих в строящееся дерево, вместе с префиксами;
(3) Шаг 2 повторяется, пока не будет достигнут искомый размер дерева (при условии, что список кандидатов, не попавших в дерево, не будет исчерпан).

Данная техника получила название sparse tree attention. Пример полученного дерева можно найти на рисунке 3. Важно отметить, что в общем случае нет гарантии, что все головы будут использованы, так что потенциально данная модификация направлена не только на решение проблемы с существенным снижением пропускной способности системы, но и позволяет незначительно снизить увеличение времени до первого токена. Однако из размеров дерева, с которым экспериментировали авторы, можно сделать вывод, что в первую очередь их исследование было нацелено на ускорение обработки запросов в однопоточном сценарии, а не на увеличение общей пропускной способности

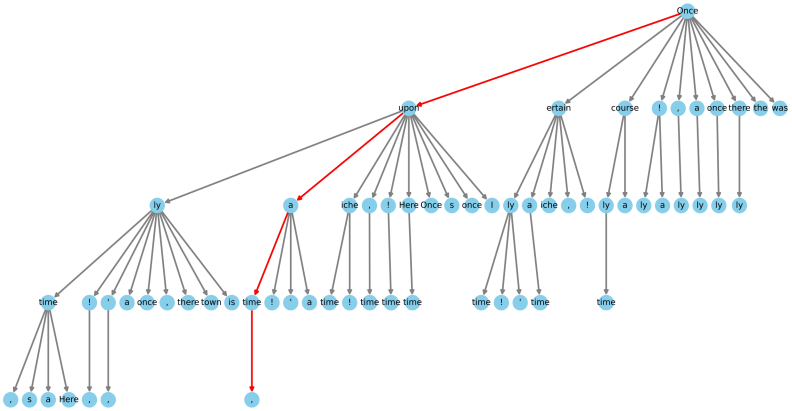

Рис. 3. Пример представления sparse tree attention в виде дерева. В первой строчке стоит токен t_{n+1}, предсказанный большой языковой моделью, в каждой из последующих строчек – токены, предсказанные соответствующим драфтером. На данном примере принятая целевой моделью последовательность полностью вошла в дерево (выделена красным).

при соблюдении SLA. Зависимость acceptance rate от числа токенов-кандидатов и изменение скорости генерации представлено на рисунке 4.

2.3. EAGLE. Авторы предлагают вернуться к авторегрессионной методологии и делают следующее наблюдение: при получении распределений q_{n+2} и последующих с помощью последнего скрытого состояния драфтовая модель не располагает достаточной информацией для того, чтобы делать уверенные предсказания в случаях, когда распределение p_{n+1} (или же распределение токена t_{n+1} по целевой модели) далеко от вырожденного, то есть даже в продолжении входной последовательности целевой моделью есть неопределенность. В качестве способа борьбы с возникающей неопределенностью предлагается представлять внутренние состояния модели для i-ой позиции в виде конкатенаций $[h_i, \mathrm{emb}(t_{i+1})]$ (emb – функция применения слоя эмбеддинга), которые подаются на вход драфтовой модели, состоящей из

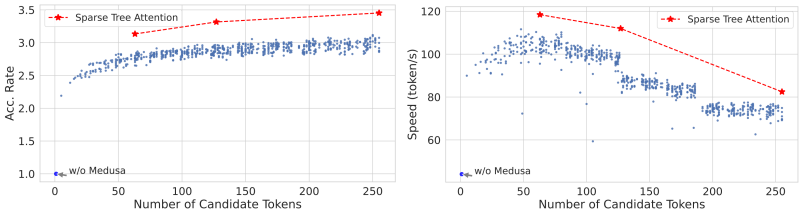

Рис. 4. Зависимость средней длины принятой последовательности (сверху) и токенов в секунду (снизу) от общего числа токенов в рассматриваемых последовательностях. Графики построены для модели Vicuna-7B, дообученной вместе с Medusa. Несмотря на рост средней длины принятого драфта, видно, что на размерах батча, с которыми работают авторы, страдает ключевая метрика числа токенов (даже при применении sparse tree attention).

одного полносвязного слоя, отображающего их в сжатое представление, и одного слоя декодера. Декодер поверх этих представлений в авторегрессионной манере генерирует последующие состояния, из которых оригинальной головой сэмплируются драфтовые токены для создания конкатенаций, необходимых для следующей итерации драфтовой модели. Аналогично Medusa, драфты генерируются в виде дерева токенов и после очередного сэмплирования для каждого драфта рассматривается несколько наиболее вероятных токенов для его продолжения. Наглядно алгоритм представлен на рисунке 5. Как видно из рисунка, особенностью данного подхода также является переиспользование компонент большой языковой модели: используются эмбеддинги токенов, которые были выучены при обучении целевой модели, и, в отличие от Medusa, голова модели также берется из целевой модели, а не обучается заново.

Поскольку для гарантии качественных предсказаний драфтера необходимо, чтобы он возвращал скрытые состояния близкие к тем, которые возвращает большая языковая модель, обучение построено на минимизации сглаженной L1-функции потерь (SmoothL1) между внутренним состоянием драфтера и значениями активаций большой

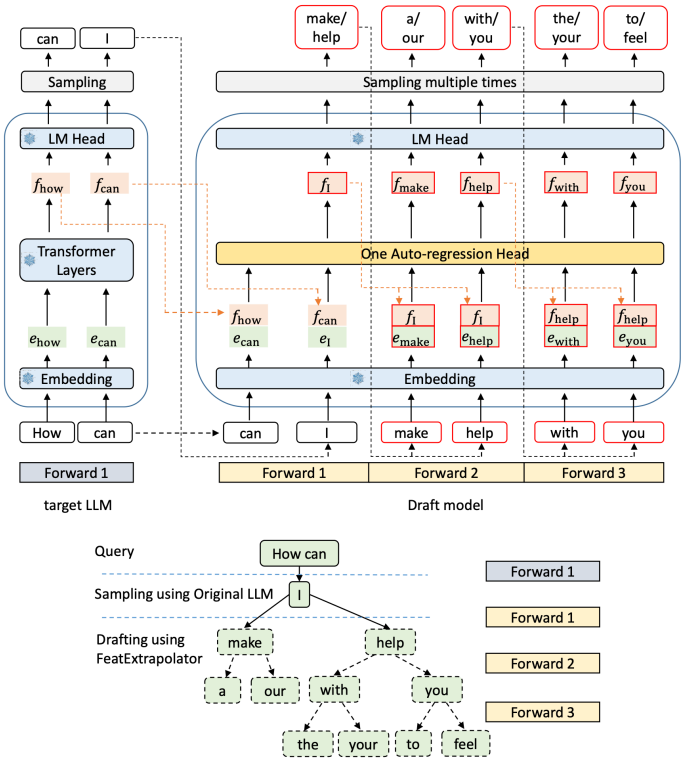

Рис. 5. Сверху: процесс применения EAGLE. При первом применении наиболее вероятный токен I используется совместно со значениями последнего слоя активации f_{can} для получения f_{I}, на основании которого сэмплируются два наиболее вероятных токена, которые используются вместе с f_{I} для предсказания следующих токенов при втором применении (третье применение на рисунке показано только для токена “help”). Синим цветом выделены все слои, взятые из целевой модели. Снизу: полное дерево кандидатов.

модели на следующем шаге. При этом, поскольку конечной целью спекулятивного декодирования является обеспечение предсказаний, консистентных с предсказаниями целевой модели, в функцию потерь с

небольшим весом добавлена кросс-энтропия между распределениями целевой и драфтовой моделей. Общая формула для функции потерь записывается следующим образом:

$$L = L_{reg} + w_{cls} \cdot L_{cls} = \text{SmoothL1}(f_{i+1}, \hat{f}_{i+1}) + \\ + w_{cls} \cdot \text{CrossEntropy}\Big(\text{Softmax}\big(\text{Head}(f_{i+1}), \text{Head}(\hat{f}_{i+1})\big)\Big), \\ \text{при } \hat{f}_{i+1} = \text{DraftModel}(T_{2:i+1}, F_{1:i}) \quad (1)$$

Здесь $F_{1:i} = (f_1, ..., f_i)$ и $T_{2:i+1} = (t_2, ..., t_{i+1})$ – набор внутренних представлений и смещенный набор токенов соответственно (из которых и получаются ранее описанные конкатенации), DraftModel – функция применения драфтовой архитектуры, возвращающая новое скрытое состояние, из которого путём применения language-modeling head получаются логиты распределения драфтового токена. Чтобы получить значение функции потерь для одной последовательности токенов из обучающей выборки функция L должна быть агрегирована по всем позициям i.

На датасете MT-Bench [9] было экспериментально установлено, что EAGLE ускоряет генерацию текста на моделях семейства Vicuna в три раза – 2.90, 3.07 и 2.95 раза на моделях Vicuna 7B, 13B и 33B соответственно (ускорение относительно других подходов показано розовым столбцом на рисунке 11 справа), тем самым обгоняя Medusa для которой те же замеры показали ускорение в 1.97, 1.92 и 1.94 раза. Однако, поскольку в EAGLE при каждом запуске спекулятивного декодирования осуществляется составление дерева потенциальных продолжений, в таком режиме метод всё ещё остается неприменимым в реальных сценариях с многопоточной обработкой запросов большой языковой моделью. Для того чтобы показать применимость и в таких сценариях, авторы показывают наличие существенного ускорения генерации без использования tree attention, о чем свидетельствует график на левой части рисунка 11. Тем не менее, для батчей большего размера в работе исследовано изменение пропускной способности только с использованием tree attention, при этом для Vicuna-7B максимальный размер батча с использованием EAGLE составил 7 при запуске на NVIDIA RTX 3090 (с 24 GB видеопамяти), а для LLaMA2-Chat 70B при запуске на четырех NVIDIA A100 (совокупно 160 GB видеопамяти) максимальный размер батча равен 4, и поэтому прирост демонстрируется только на размере батча до 4 для обеих моделей (см. таблицу 2).

Batch size	1	2	3	4	Throughput
Vicuna 7B	2.90x	2.87x	2.65x	2.76x	1.97x
LLaMA2-Chat 70B	3.01x	2.81x	2.50x	2.40x	1.99x

Таблица 2. Изменение скорости генерации и пропускной способности модели (throughput) с использованием EAGLE на батчах малого размера. Значение throughput выбрано как максимальное из всех протестированных размеров батча.

2.4. EAGLE-2. Данная модификация [5] строится вокруг следующего наблюдения: несмотря на то, что tree attention в первой версии позволяет увеличить скорость генерации, подбор структуры этого дерева является трудоемким процессом, зависящим от запросов, под которые обучается драфтовая модель, а само дерево не адаптивно, поскольку даже в рамках одного сценария применения распределение входных запросов может изменяться и, как следствие, наиболее эффективное подмножество проверяемых драфтов – вместе с ним. В качестве гипотезы выдвигается предположение о том, что архитектуру дерева можно подбирать в процессе применения драфтовой модели, путём сбора оценки acceptance rate для каждого токена в дереве и подбором архитектуры, максимизирующей суммарную оценку. Обновление дерева в режиме онлайн является главным отличим предлагаемого подхода от реализации sparse tree attention в Medusa, в которой статистики по acceptance rate для каждого токена в дереве считаются один раз по валидационному датасету.

Поскольку acceptance rate является метрикой, которую нельзя получить перед применением целевой модели, в качестве её оценки для токена используется уверенность (confidence score) драфтовой модели в этом токене, то есть вероятность, полученная в результате применения функции Softmax поверх логитов, возвращённых головой драфтовой модели. Авторы экспериментально установили наличие сильной линейной корреляции между этими двумя показателями, что обосновывает применение оценки через confidence score. При этом вероятностный характер confidence score позволяет использовать мультипликативную оценку acceptance rate для токенов в драфте, ищущих после

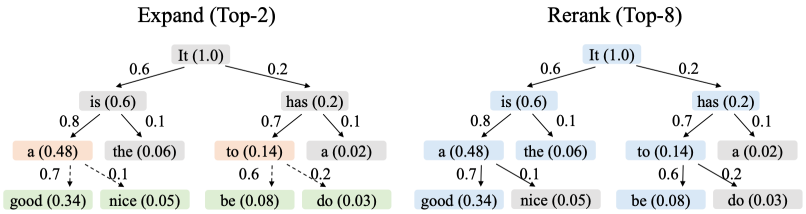

Рис. 6. Во время фазы расширения (сверху) серые и оранжевые позиции уже находятся в дереве, при этом оранжевым позициям соответствует $s = 2$ токенов с наилучшей оценкой A_i среди токенов на последнем слое; на этих токенах (и соответствующих им скрытых состояниях) драфтовая модель запускается ещё раз и выводятся новые оценки уверенности, обозначенные зеленым цветом. На фазе переранжирования (снизу) учитываются все позиции, включая новые, и выбирается $m = 8$ позиций с токенами, из которых будут составлены драфты, отправленные на верификацию целевой моделью.

первого, через произведение confidence score для всех токенов в префиксе, а именно

$$A_i = \prod_{t_j \in \mathrm{Path}(root, t_i)} a_j \approx \prod_{t_j \in \mathrm{Path}(root, t_i)} c_j \tag{2}$$

Где a_j – обозначение acceptance rate, также используемое в разделе 2.3, c_j – confidence score для токена, $root$ – обозначение вершины в дереве, соответствующей токену t_{n+1}. Обновление дерева T с использованием оценки уверенности 2 происходит следующим образом (также проиллюстрировано на рисунке 6):

(1) Для всех позиций t_i (не путать с обозначением для токенов в последовательности) из текущего дерева T осуществляется применение драфтовой модели таким же образом, как и в подходе EAGLE и для них считаются соответствующие оценки A_i;
(2) Фаза расширения (expansion phase): после последнего применения драфтовой модели рассматривается s наибольших A_i по всем позициям на последнем слое из текущего дерева (которые

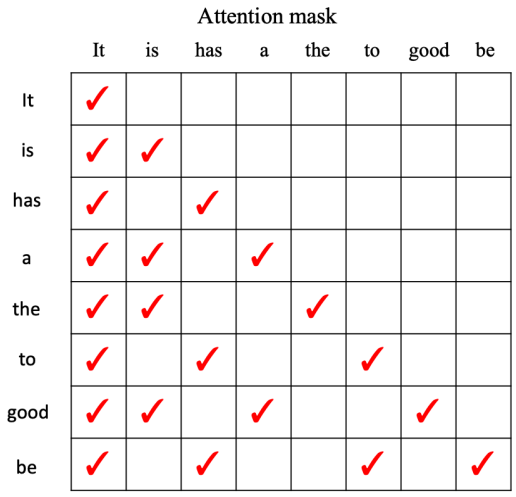

Рис. 7. Древовидная маска внимания, учитывающая структуру дерева, которое было получено после фазы переранжирования на примере из рисунка 6.

будут обозначены $T' = \{t'_i\}_{i=1}^{s}$), далее драфтовая модель применяется ещё раз к токенам, соответствующим этим позициям, и выводятся оценки уверенности $\{A'_{i,j}\}_{i=1,j=1}^{s,|V|}$ для всех токенов, при продолжении ими последовательностей, соответствующих токенам из T';

(3) Фаза переранжирования (reranking phase): из $\{A_i\} \cup \{A'_{i,j}\}$ выбирается m наибольших значений и соответствующие им токены формируют драфтовые последовательности, которые будут верифицированы целевой моделью (для этого необходимо, чтобы новое множество вершин было связным деревом, а истинность этого утверждения следует из определения A_i – оценка для вершины-предка всегда будет не меньше оценки для любой из его вершин-потомков) с использованием tree attention с маской, построенной по структуре дерева драфтовых последовательностей (пример изображен на рисунке 7); полученное дерево становится новым деревом T.

Сама архитектура драфтера и способ её применения остались неизменными. В экспериментальной части так же, как и в первой версии подхода, в первую очередь рассматривается прирост числа токенов в секунду в однопоточном сценарии, и сообщается, что EAGLE-2 демонстрирует прирост порядка 20-40% относительно EAGLE. На моделях из серии Vicuna наибольший прирост на бенчмарке MT-Bench был показан на модели Vicuna 13B и равен 4.26 раза (изображён зеленым столбцом на правой части рисунка 11). В контексте мультиарендных сервисов, однако, подобная надстройка над EAGLE не представляет такой же ценности, поскольку преимущество от её использования зависит от готовности выделить батч достаточного размера под каждый запрос, что не представляется возможным в высоконагруженных сценариях.

2.5. EAGLE-3. Далее ставится следующая проблема реализаций EAGLE: использование функции потерь SmoothL1 на факторах в процессе обучения заставляет драфтовую модель работать с одним внутренним представлением целевой модели и не позволяет учитывать более емкий сигнал, который можно получить из совокупности низкоуровневых (low-level), среднеуровневых (mid-level) и высокоуровневых (high-level) факторов, вычисляемых моделью, т.е. из скрытых состояний с нескольких уровней сразу.

В EAGLE-3 [6] в качестве решения данной проблемы предлагается обучать модель только на минимизацию дивергенции между распределением на токенах, предсказываемых драфтовой моделью, и распределением токенов по целевой модели, однако наивное исключение SmoothL1-слагаемого из функции потерь приводит к переобучению скрытого состояния, поддерживаемого внутри драфтовой модели, под задачу предсказания следующего токена, вследствие чего качество драфтов, начиная со второго токена, становится ещё хуже, чем при использовании SmoothL1 во время обучения (см. рисунок 8).

С целью предотвращения переобучения и увеличения репрезентативности скрытого состояния, поддерживаемого в драфтовой модели, вводится процедура training-time test. Данная процедура заключается в том, что на каждом этапе обучения перед подсчётом градиентов делается не один прямой проход (forward pass) через драфтовую модель, а γ проходов, и градиенты считаются на сумме дивергенций распределений с коэффициентами дисконтирования, геометрически убывающими с увеличением номера прохода (для приоритизации увеличения

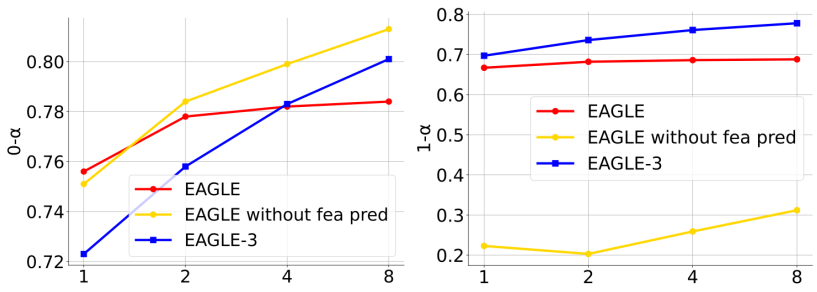

Рис. 8. Доля принятий первого токена в потенциальной последовательности (0-α) и второго токена (1-α) относительно числа кандидатов. Видно, что подход с обучением на вероятность следующего токена без дополнительных изменений (EAGLE without fea pred) негативно сказывается на способности драфтовой модели генерировать качественные последовательности длиннее одного токена.

качества предсказаний более ранних токенов в драфте над более поздними). Процесс training-time test схематично представлен на рисунке 9.

Взяв во внимание выражение (1) для функции потерь при обучении EAGLE, заметим, что новая функция потерь будет состоять из нескольких слагаемых, соответствующих вычислению кросс-энтропии между распределениями целевой и драфтовой моделей, тем самым принимая следующий вид:

$$L = \sum_{k=1}^{\gamma} \text{CrossEntropy}\Big(\text{Softmax}\big(\text{Head}(f_{i+k}), \text{Head}(\hat{f}_{i+k})\big)\Big)$$

Значения $(f_{i+1}, ..., f_{i+\gamma})$ соответствуют применению целевой модели на последовательности $(T_{1:i+1}, T'_{i+2:i+\gamma+1})$, однако, аналогично предыдущей функции потерь, значение L необходимо далее агрегировать по всем i, поэтому при обучении драфтовой модели на предсказание драфта из γ токенов необходимо уметь делать прямой проход целевой модели на всех последовательностях из множества $\{(T_{1:i+1}, T'_{i+2:i+\gamma+1})\}_{i=1}^{n}$, т.е. на продолжениях длины γ относительно каждого префикса последовательности токенов из обучающей выборки. Для обеспечения одновременного применения целевой модели

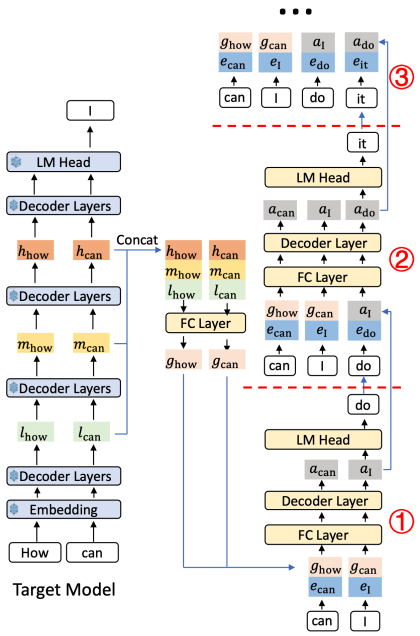

Рис. 9. Процедура выполнения трёх шагов training-time test (предсказанные токены выписаны для первых двух шагов). Состояния целевой модели с разных уровней конкатенируются и пропускаются через полносвязный слой, и поверх них запускается драфтовая модель. Скрытые состояния $a_{(\cdot)}$ (в EAGLE были обозначены как $\hat{f}_{(\cdot)}$) и эмбеддинги токенов из формируемого драфта используются наряду со скрытыми состояниями целевой модели $g_{(\cdot)}$ ($f_{(\cdot)}$ в EAGLE) и сэмплированными из них токенами аналогично тому, как это делается в предыдущих версиях EAGLE.

ко всем последовательностям маска внимания модифицируется (рисунок 10) и увеличивается в γ раз по одной из размерностей, что увеличивает требования к объёму видеопамяти на машине, на которой производится обучение драфтера. На процесс применения драфтовой модели данная модификация никак не влияет.

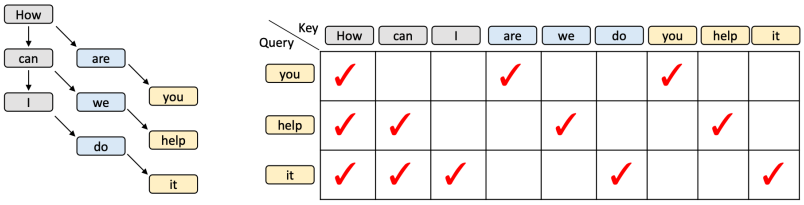

Рис. 10. Модифицированная во время training-time test маска внимания целевой модели для проверки продолжений на $\gamma = 2$ токена относительно каждого префикса входной последовательности “How can I” одновременно. Синие токены соответствуют первым токенам в драфтах для каждого префикса, а желтые – вторым токенам. Для входной последовательности длины N размер маски составляет $\gamma \cdot N \times N$.

Batch size	2	4	8	16	24	32	48	56
EAGLE	1.30x	1.25x	1.21x	1.10x	1.03x	0.93x	0.82x	0.71x
EAGLE-3	1.75x	1.68x	1.58x	1.49x	1.42x	1.36x	1.21x	1.01x

Таблица 3. Сравнение числа токенов в секунду на батчах разного размера на движке VLLM.

Экспериментально в статье показано ускорение генерации текста в однопоточном сценарии в более, чем в пять раз, в частности, ускорение на модели Vicuna 13B на MT-Bench составило 5.58 раза (изображено фиолетовым столбцом справа на рисунке 11).

Стоит отметить, что, в отличие от EAGLE-2, предлагаемые в EAGLE-3 модификации не опираются на tree attention в процессе применения, что делает их применимыми и в сценариях с множеством параллельных запросов. Более того, авторы провели эксперименты с запуском модели с EAGLE-3, а также EAGLE, на батчах большого размера и на движке VLLM им удалось показать наличие прироста в скорости генерации. Результаты показаны в таблице 3.

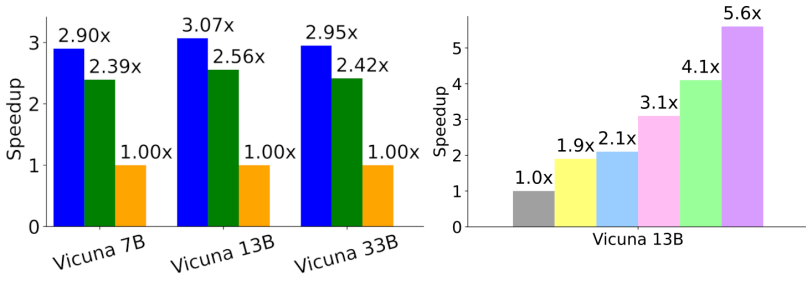

Рис. 11. Слева: сравнение числа токенов в секунду с использованием tree attention (синий столбец) и без него (зеленый столбец) относительно использования целевой модели без спекулятивного декодирования на датасете MT-Bench. Справа: сравнение различных подходов спекулятивного декодирования с точки зрения прироста числа токенов в секунду для Vicuna-13B на бенчмарке MT-Bench. Подходы указаны слева направо в следующем порядке: без спекулятивного декодирования, оригинальный подход, Medusa, EAGLE, EAGLE-2, EAGLE-3.

2.6. ReDrafter. В данном подходе так же, как и в EAGLE, предлагается применять авторегрессионную генерацию при спекулятивном декодировании, однако в качестве самой драфтовой модели используется рекуррентная нейронная сеть (recurrent neural network, RNN). Авторы критикуют архитектуру Medusa, поскольку при её эффективности и способности к параллельному исполнению она совсем не учитывает последовательную природу текстовой генерации. При этом RNN является менее ресурсоемкой архитектурой в сравнении с блоком декодера, который фактически используется в EAGLE.

В ReDrafter [1], аналогично Medusa, в качестве исходного скрытого состояния модели используются активации последнего скрытого слоя большой языковой модели, и при помощи лучевого поиска (beam search) [2] выбирается w наиболее вероятных продолжений длины γ. Алгоритм построения драфтов состоит из следующих шагов:

(1) На первом шаге на вход драфтеру подается эмбеддинг токена t_{n+1}, сэмплированного из распределения целевой модели, а

в качестве первоначального скрытого состояния используется h_n – последнее скрытое состояние целевой модели; на выходе получается пара из нового скрытого состояния h_{n+1} и логитов драфтового распределения q_{n+2}, данная пара из скрытого состояния и распределения сохраняется в памяти;

(2) На втором шаге драфтер рассматривает новое скрытое состояние h_{n+1} и w наиболее вероятных токенов, или же драфтов единичной длины, по распределению q_{n+2} и запускается параллельно на парах из скрытого состояния и токена вида $\{(h_{n+1}, t'^k_{n+2})\}_{k=1}^w$, где верхний индекс k означает порядковый номер по убыванию вероятности сэмплирующего распределения, и получается w пар $\{(h^k_{n+2}, q^k_{n+3})\}_{k=1}^w$ (заметим, что после этого шага будет необходимо параллельно работать уже с несколькими скрытыми состояниями драфтовой модели);

(3) На i-ом шаге ($3 \leqslant i \leqslant \gamma$) на вход драфтеру подаётся w пар $\{(h_{n+i-1}, T'^k_{n+2:n+i})\}_{k=1}^w$ с предыдущего шага, где T' представляют собой драфты с наибольшей совокупной вероятностью, то есть для которых максимизируется величина Q, вычисляемая динамически по формуле:

$$Q(T'_{n+2:n+i}) = Q(T'_{n+2:n+i-1}) \cdot q^k_{n+i}(t'_{n+i}|T'_{n+2:n+i-1}),$$
$$\text{при } Q(T'_{n+2:n+2}) = q_{n+2}(t_{n+2}) \tag{3}$$

(При этом k всё ещё означает порядковый номер драфта, но теперь уже по совокупной вероятности Q).

Так же, как и для других методов спекулятивного декодирования, для ReDrafter свойственно наличие у драфтовых последовательностей общего префикса, и для того, чтобы целевую модель не пришлось запускать на множестве последовательностей с одинаковыми префиксами (с множеством дублирующихся вычислений), производится маскирование повторяющихся токенов в драфтах и построение модифицированной маски внимания при помощи процедуры packed beam attention, которая проиллюстрирована на рисунке 12.

Вместо кросс-энтропии, используемой в методах Medusa и EAGLE при обучении драфтовой модели, обучение рекуррентной сети в ReDrafter производится на логарифм правдоподобия токенов в драфте

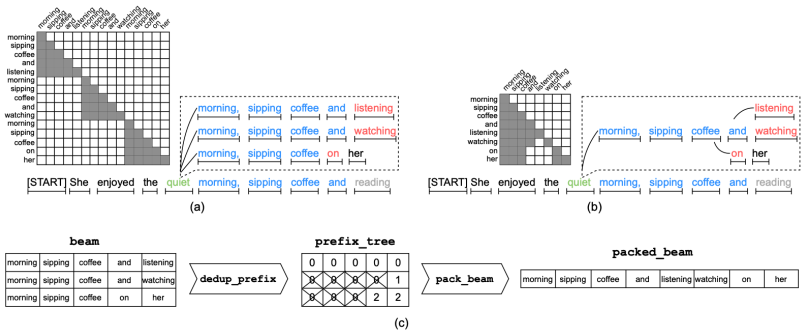

Рис. 12. Процесс применения packed beam attention. На рисунке (a) изображена наивная маска внимания для трех последовательностей, на рисунке (b) изображена сжатая маска с убранными дублирующимися префиксами. На рисунке (c) показаны этапы процесса: операция dedup_prefix маскирует повторяющиеся префиксы, а операция packed_beam упаковывает оставшиеся токены в общую последовательность, основываясь на расположении незамаскированных токенов.

по целевой модели. Иными словами, минимизируется значение следующей функции потерь:

$$L_{\text{distill}} = -\sum_{n} \log P(T'_{n+2:n+\gamma+1} | T_{1:n+1})$$

$$= -\sum_{n} \left(\log p(t'_{n+2} | T_{1:n+1}) + \sum_{i=2}^{\gamma} \log p(t'_{n+i+1} | T_{1:n+1}, T'_{n+2:n+i}) \right)$$

Несмотря на более простую архитектуру, авторам данного подхода удалось обойти Medusa и сравниться с EAGLE: приросты скорости генерации составили 2.4, 2.74 и 2.8 раз для Medusa, EAGLE и ReDrafter соответственно на целевой модели Vicuna-13B на бенчмарке MT-Bench.

Более емкой частью результатов является исследование зависимости скорости генерации, а также пропускной способности модели, в зависимости от изменения параметров ширины луча и размера батча запросов, обрабатываемых параллельно. Значения представлены в

BSZ	BW=1		BW=2		BW=4		BW=8		BW=16		BW=32		BW=64	
	TPS	TPS×BSZ	TPS	TPS×BSZ	TPS	TPS×BSZ	TPS	TPS×BSZ	TPS	TPS×BSZ	TPS	TPS×BSZ	TPS	TPS×BSZ
1	62.55	62.55	73.32	73.32	80.47	80.47	88.31	88.31	100.16	100.16	104.51	104.51	**110.64**	110.64
2	58.56	117.13	70.25	140.49	75.77	151.54	86.64	173.29	99.45	198.90	107.52	215.04	**111.42**	222.83
4	57.51	230.03	67.42	269.67	74.77	299.09	81.95	327.81	97.13	388.51	**99.11**	396.46	85.23	340.94
8	53.14	425.14	60.48	483.85	69.49	555.90	81.14	649.11	**83.44**	667.52	73.41	587.30	54.19	433.48
10	53.61	536.14	58.79	587.85	66.29	662.88	71.55	715.48	**73.57**	735.68	65.31	653.10	44.23	442.25
20	45.16	903.20	49.05	980.95	**57.35**	1146.95	53.87	1077.45	48.76	975.14	35.66	713.17	23.28	465.70
40	32.69	1307.65	33.59	1343.41	**36.68**	1467.04	34.33	1373.08	26.43	1057.20	19.15	766.03	12.09	483.58
80	18.94	1515.40	**20.45**	1636.36	19.65	1571.89	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM

Таблица 4. Сравнение скорости генерации, выраженной в числе токенов в секунду (TPS), а также пропускной способности, выраженной в размере батча, умноженном на среднее число токенов в секунду (TPS × BSZ), для различных значений ширины луча и размера батча. Указаны результаты работы Vicuna-7B на бенчмарке MT-Bench.

таблице 4. Из неё можно сделать вывод, что подход легко конфигурируется под разные нужды: в рамках систем, от которых ожидается максимально быстрый ответ и в которые отправляется единичное число запросов, более осмысленно использовать лучевой поиск с большей шириной, в то время как при высокой нагрузке излишне большое значение ширины луча может негативно сказаться на скорости ответа и пропускной способности в целом.

2.7. Online Speculative Decoding. Существенная проблема, которая не решается ни в одном из предыдущих подходов, состоит в том, что модель спекулятивного декодирования, как и прочие модели машинного обучения, подвержена деградации вследствие сдвига в распределении входных данных (data shift). Поскольку распределение запросов меняется со временем, меняется и распределение ответов, вследствие чего новое распределение хуже приближается драфтовой моделью. Один из способов решения данной проблемы предлагается в виде алгоритма дообучения драфтовой модели по мере её применения.

В своей работе [7] авторы основываются на исходной реализации спекулятивного декодирования и внедряют процесс, отвечающий за сохранение последовательностей, на которых драфтовая модель сделала ошибку, и осуществляющий её дообучение путём дистилляции знаний из целевой модели. Эксперименты проводятся с Vicuna-7B с LLaMA2-160M в качестве драфтовой модели (архитектуры, к которым

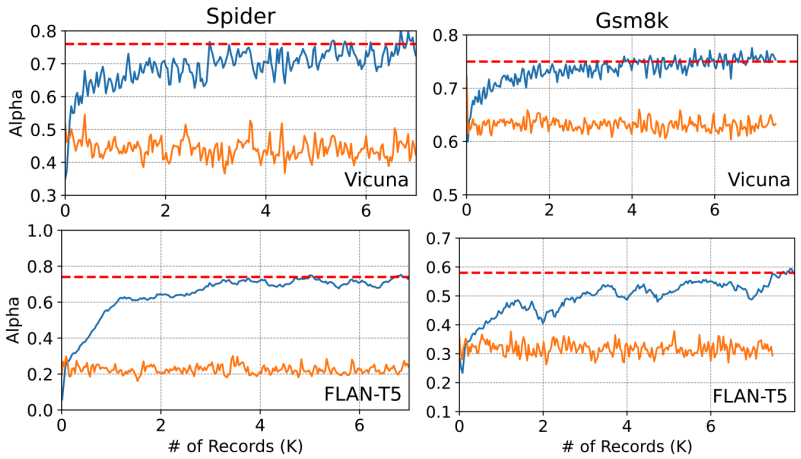

Рис. 13. Изменение acceptance rate драфтовой модели по мере обработки тестового датасета. Изображены экспоненциально взвешенные скользящие средние α для Online Speculative Decoding (синий график) и для модели, предобученной на 10% данных (оранжевый график). Красная линия показывает средний acceptance rate на всем датасете для Online Speculative Decoding.

часто возвращались в серии статей про EAGLE), а также FLAN-T5-XXL с FLAN-T5-Small в качестве драфтовой модели (рассматривались в первой статье про спекулятивное декодирование). На рисунке 13 приведены графики изменения acceptance rate для модели с дообучением в режиме онлайн, а также для модели, обученной предварительно на 10% потока (для симуляции обработки новых запросов моделью, обученной на данных, собранных к фиксированному моменту времени). Из графиков видно, что предварительно обученная модель в среднем одинаково показывает себя на всем датасете, а онлайн-обучение позволяет быстро увеличить долю принятий и приблизить её к показателям модели, обученной на всех тренировочных данных.

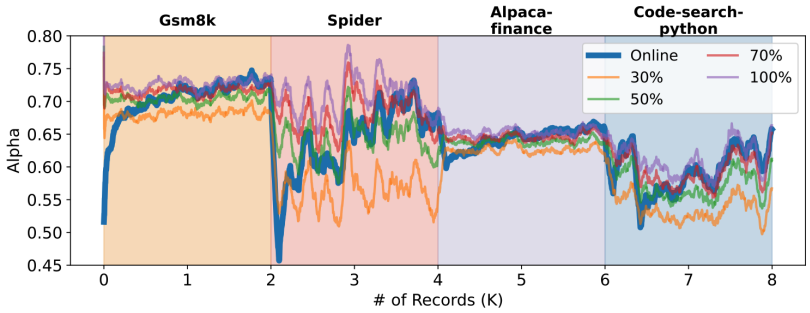

Рис. 14. Изменение acceptance rate драфтовой модели при работе на потоках из разных датасетов. Отображены значения для Online Speculative Decoding, а также для моделей, обученных в оффлайне на различных долях данных.

При этом в случае резкой смены потока наблюдается существенное ухудшение системы с точки зрения доли принятых токенов от драфтовой модели (см. рисунок 14), однако отмечается, что по мере обработки запросов из нового датасета качество возвращается на уровень моделей, обученных на 70% и на 100% имеющихся данных.

Несомненно, в случае рассмотрения одного часто меняющегося потока запросов, внедрение данной модификации позволит адаптировать драфтовую модель и поможет избежать уменьшения скорости обработки потока. Однако такой сценарий применения плохо моделирует действительную схему взаимодействия с мультиарендным сервисом, использующим большую языковую модель, поскольку на деле, как правило, сервис одновременно работает с несколькими потоками запросов, приходящих от разных клиентов. Таким образом, Online Speculative Decoding, хоть и является адаптирующимся в реальном времени методом, всё ещё направлен на интегральное улучшение системы и не способен одновременно учитывать потоки запросов, идущие от разных клиентов, и улучшать скорость работы независимо для каждого из них. Также важно учесть, что нагрузка, вызываемая дообучением модели с помощью предлагаемого алгоритма не позволяет назвать его “онлайновым”, поскольку при его исполнении необходимо

выделять мощности под пересчёт состояний целевой и драфтовой моделей и обучение на собранных запросах, что вводит дополнительные накладные расходы при эксплуатации системы, а также потенциальные задержки при обработке запросов (часть машин будет необходимо выделить под дообучение модели).

Метод, предлагаемый в следующем разделе, направлен на улучшение спекулятивного декодирования в мультиарендном сценарии. Данный метод позволяет ввести в систему учёт генераций для каждого клиента по отдельности, что позволяет драфтовой модели лучше адаптироваться к разным клиентам одновременно и увеличивает совокупную пропускную способность системы.

§3. Метод: Client-Conditioned Speculative Decoding (CCSD)

В данном разделе описывается разработанный поверх ReDrafter метод CCSD (Client-Conditioned Speculative Decoding), направленный на увеличение пропускной способности системы, применяющей спекулятивное декодирование путем использования данных о клиентах, отправляющих запросы в систему, а также времени отправления запроса.

3.1. Обоснование предлагаемого метода. Поскольку спекулятивное декодирование предполагает, что целевая модель остаётся неизменной, нетрудно видеть, что в общем алгоритме, описанном в разделе 2.1, шаг 2 является единственным, который можно модифицировать (использование других алгоритмов сэмплирования и релевантные эксперименты остаются за рамками данной статьи). При этом, если целью ставится ускорение работы системы с большой языковой моделью, модификация драфтовой модели, применяемой на данном шаге, должна повлечь увеличение сходства возвращаемых ей распределений с распределениями, из которых целевая модель сэмплирует токены. В мультиарендном сценарии каждый клиент ожидает получить ответ от модели на заведомо узком сценарии, определяемом как типом запросов, отправляемых клиентом, так и ожидаемым форматом ответа. По этой причине имеет смысл снабдить алгоритм спекулятивного декодирования некоторым уникальным для клиента априорным знанием о том, что он ожидает увидеть в качестве ответа на свой запрос.

Чтобы рассмотреть метод внедрения подобного априорного знания, заметим, что выражение (3), значение которого максимизируется при лучевом поиске, представляет собой рекуррентную формулу, в которой на каждой итерации происходит домножение на вероятность токена по драфтовой модели. Значение вероятности токена можно заменить на адаптивную вариацию, обусловливающуюся не только на предыдущие токены в последовательности, но и на данные о предыдущих генерациях клиента. Воспользуемся идеей формулы Байеса и введем априорное распределение, которому будет делегирован учет клиента и времени генерации (с целью упрощения записи индексы для q не указаны):

$$q(t'_{n+i+1}|T'_{n+2:n+i}, id, ts) \propto q(t'_{n+i+1}|T'_{n+2:n+i}) \cdot q_{client}(T'_{n+2:n+i}, t'_{n+i+1}|id, ts)$$

Учёт параметра ts (timestamp, временная метка) помимо идентификатора клиента id является критически важным условием, поскольку в рамках каждого клиента поток запросов все ещё может изменяться, и для обеспечивания наибольшего сходства с действительностью q_{client} должен обновляться “на лету”. Помимо этого работа поверх генераций в живом времени означает, что введенное априорное распределение будет способно учитывать и запросы, которых не было в обучающей выборке ввиду того, что драфтовая модель обучалась на более ранних данных, и она может не обладать достаточно актуальным представлением о клиенте.

В качестве легковесной модели клиента, способной обновляться в онлайн-режиме без оказания существенного влияния на производительность системы была выбрана n-граммная модель. Несмотря на её относительную простоту (по сравнению с той же драфтовой моделью, выраженной в виде рекуррентной нейронной сети), такой выбор обусловлен спецификой задачи мультиарендности и необходимостью динамической адаптации. Помимо своего быстродействия n-граммы позволяют хранить индивидуальные легковесные модели для большого числа клиентов. Предполагается, что подобный экспертный сигнал по недавней истории клиента может скорректировать предсказания даже более сложной модели в нужную сторону, особенно для более частотных, а также шаблонных запросов клиента.

С точки зрения обучения n-граммная модель представляет собой одну из простейших языковых моделей:

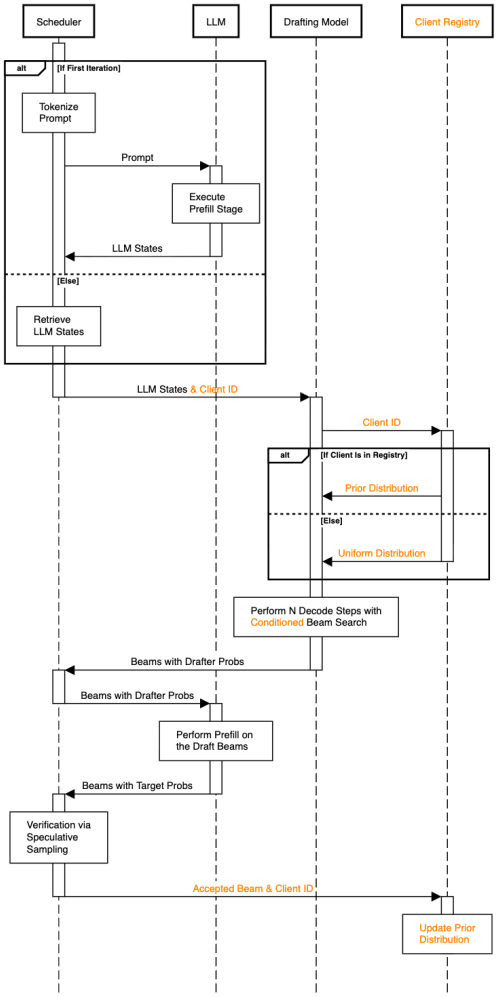

Рис. 15. Диаграмма последовательностей, изображающая одну итерацию спекулятивного декодирования. Оранжевым цветом выделены сущности и шаги, добавленные при внедрении CCSD.

- входные данные представляют собой набор текстов, каждый из которых токенизируется;
- за один проход по каждому токенизированному запросу рассматриваются все кортежи длины n состоящие из последовательных токенов, то есть кортежи вида $(t_{i-n+1}, t_{i-1}, t_i)$ (при $i < n$ первые позиции заполняются системным beginning-of-sentence токеном);
- для каждого кортежа обновляется соответствующая n-граммная статистика: счетчик вида $\#\{t_i | t_{i-n+1}, ..., t_{i-1}\}$ – количество раз, когда токен t_i встретился после $n-1$ определенных токенов (в случае униграммных статистик обновляются безусловные счетчики $\#\{t_i\}$).

Возможность онлайнового обновления очевидна: после каждого шага спекулятивного декодирования принятую последовательность можно разбить на n-граммы и обновить статистики, что занимает ничтожно малое время по сравнению с применением драфтовой модели.

Общий алгоритм спекулятивного декодирования вместе с предлагаемыми модификациями изображен в виде диаграммы последовательностей на рисунке 15.

3.2. Реализация n-граммных статистик. Начнём с униграммной модели: для одного клиента ее легко поддерживать в виде вектора размера $|V|$ (здесь и далее V обозначает словарь токенов, с которым работает целевая модель), содержащего целые числа – количество раз, которые токен был сгенерирован при ответе на запрос. Для каждого клиента нужен свой вектор встречаемостей, поэтому проще хранить матрицу размера $|C| \cdot |V|$, где C – множество поддерживаемых клиентов. Несмотря на то, что размеры словарей в современных больших языковых моделях достигают сотен тысяч, например, порядка 128k токенов для моделей семейства LLaMA 3, всю данную матрицу можно хранить в видеопамяти даже на одной видеокарте (для одного клиента получится размер $2^7 \cdot 10^3 \cdot 4$ байт $\approx 2^{19}$ байт $= 512$ KiB). Далее, при необходимости получить априорное распределение, достаточно взять вектор для нужного клиента, поделенный на его сумму значений в нём.

К сожалению, такой же наивный подход для биграммной модели реализовать не получится, поскольку в таком случае нужно будет хранить в памяти матрицу размера $|C| \cdot |V|^2$, и новая оценка памяти даже для одного клиента будет равна $(2^7 \cdot 10^3)^2 \cdot 4$ байт $= 2^{16} \cdot 10^6$ байт ≈ 65.5

GB, что является чрезмерно большим объёмом даже для современных видеокарт, предназначенных для работы с нейронными сетями, например, A100 и H100, имеющими 80 GB видеопамяти.

Однако, если рассмотреть биграммные статистики, получаемые при работе с реальными данными, для отдельных клиентов, окажется, что матрицы биграмм будут очень разреженными, а для некоторых токенов в них вообще не будет статистик, поскольку эти токены не встречались в процессе генерации, например, если модель обрабатывает запросы от пользователей из России, в большинстве случаев китайские иероглифы при ответе на них генерироваться не будут. Следовательно, для хранения матриц биграмм следует использовать разреженные матрицы. Экспериментально удалось установить, что после обработки 500 тысяч запросов с одного потока, в матрице в координатном формате хранится порядка 700 тысяч ненулевых элементов, что позволяет теоретически оценить ее размер в памяти, как $3 \cdot 7 \cdot 10^5 \cdot 4$ байт $= 84 \cdot 10^5$ байт $= 8.4$ MB (хранение “троек” из номера строки, номера столбца и значения в виде четырёхбайтовых целочисленных значений). При загрузке матриц для каждого клиента в координатный формат на фреймворке PyTorch у каждой матрицы наблюдается размер порядка 20-30 MiB, что является вполне приемлемым значением, позволяющим применить биграммный метод на практике.

§4. Результаты

4.1. Постановка экспериментов. Для проведения экспериментов с логов запросов клиентов Yandex Cloud были собраны два датасета – обучающий и тестовый. Обучающий датасет собран из запросов с июля по декабрь 2024 года, в то время как тестовый датасет собран за период с января по май 2025 года для того, чтобы просимулировать внедрение модели в сервис в начале 2025 года: модель обучается только на тех данных, которые имелись к этому моменту, и ее качество оценивается на новых запросах, которые не могли попасть в обучающий датасет непосредственно. В качестве целевой модели была взята модель YandexGPT Lite 5 с 8 миллиардами параметров.

Тестовые данные агрегированы по дням, с целью ускорения проводимых экспериментов выбраны дни с периодичностью в одну неделю, и из оставшихся элементов сконструировано 5 датасетов: один совокупный и 4 срезовых. В совокупном датасете с каждого из оставшихся

дней была взята выборка размера 1000 запросов, а для каждого из срезовых датасетов было взято 200 запросов от одного из клиентов.

В качестве базового подхода, относительно которого будут оцениваться методы, предлагаемые в данной статье, предлагается рассмотреть рекуррентную сеть, обученную с параметрами обучения, указанными в исходной реализации. Это делается для того, чтобы наиболее соответствовать экспериментам, которые были осуществлены авторами ReDrafter. Датасет собран путем выбора 500 тысяч случайных запросов из множества запросов таких, что сумма длин токенизированного запроса и ответа, сгенерированного YandexGPT, не превышает 2048 токенов. Данный параметр был взят из исходной реализации, и он позволяет проводить обучение с размером батча 4, что существенно уменьшает время обучения с сохранением его стабильности.

Тестовый подход представляет собой использование той же самой модели для осуществления спекулятивного декодирования, но обусловленный на априорные распределения клиентов, о которых говорилось в предыдущем разделе. Рассмотрены униграммная и биграммная модели априорного распределения. Если статистики не посчитаны для клиента или клиент не содержится в множестве, для которого система считает статистики (как правило, клиенты, отправляющие чрезмерно малое количество запросов в систему), априорное распределение для него считается равномерным и, таким образом, внедренный метод не будет оказывать влияние на распределение вероятностей токенов при генерации кандидатов для данного клиента. Несмотря на то, что модель спекулятивного декодирования не обучается на тестовых данных, тестовые данные учитываются при пересчете n-граммных статистик для клиента и влияют на то, какие драфты будут выбираться на запросах, собранных с последующих дней.

4.2. Результаты экспериментов и их анализ. На рисунке 16 изображено изменение acceptance rate, а также числа генерируемых токенов в секунду, при внедрении обусловливания на униграммное априорное распределение по клиентам на совокупном потоке. К сожалению, при внедрении униграммного подхода улучшить систему с точки зрения целевых метрик не удалось, и можно сделать вывод о том, что в униграммной модели содержится недостаточное количество информации для улучшения предсказаний. Графики, соответствующие биграммному априорному распределению, показаны на рисунке 17, и из них можно сделать вывод о наличии положтельного влияния на

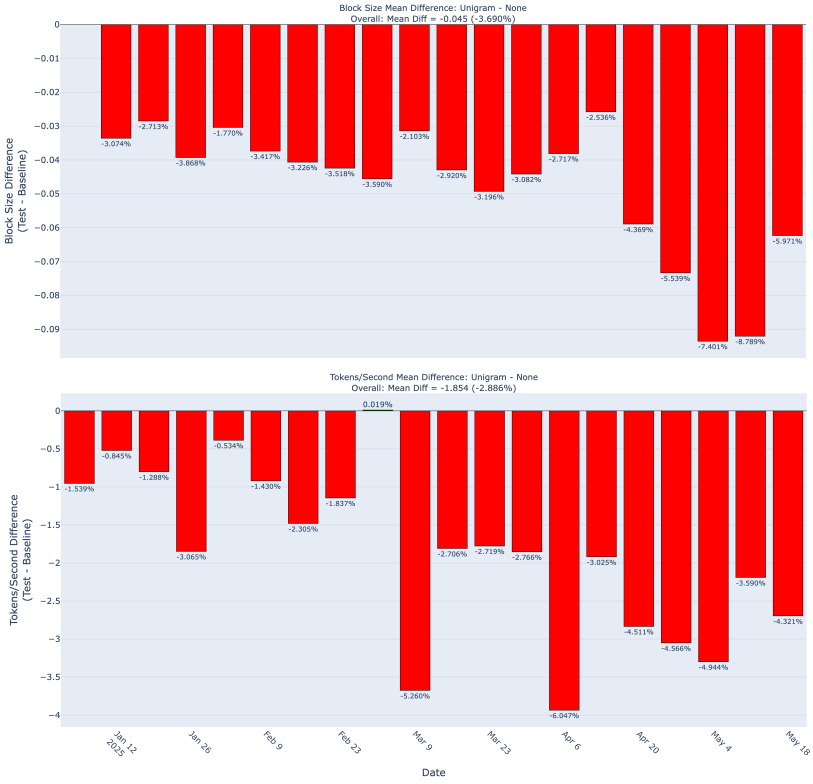

Рис. 16. Разница (абсолютная и относительная) метрик спекулятивного декодирования при использовании униграммного априорного распределения и без его использования (соответственно, test и baseline) на совокупном тестовом датасете.

качество драфтов с теоретической точки зрения: acceptance rate вырос на величину до $+5\%$ или же 0.083 токена за один шаг спекулятивного декодирования. Полученный прирост является статистически значимым, однако, с другой стороны, улучшить показатель числа токенов, генерируемых в секунду, путем использования биграммной модели априорного распределения не удалось.

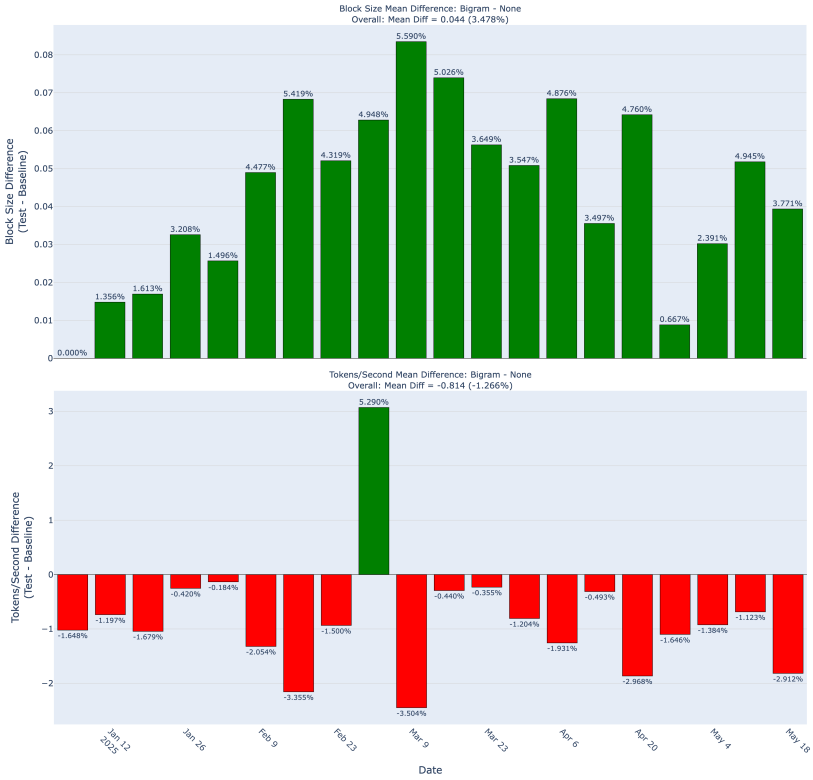

Рис. 17. Разница (абсолютная и относительная) метрик спекулятивного декодирования при использовании биграммного априорного распределения и без его использования (соответственно, test и baseline) на совокупном тестовом датасете.

Причиной отсутствия прироста числа токенов в секунду служат вычислительные издержки текущей реализации и необходимость в инженерных модификациях кода вычисления априорных распределений для клиентов. Экспериментально установлено, что при текущей реализации каждая итерация beam search занимает на $+25\%$ больше времени, а итоговое время, необходимое на одну итерацию спекулятивного декодирования, составляет на $+5\%$ больше.

Далее проверено наличие прироста от спекулятивного декодирования в узких сценариях: эксперимент был повторен в том же формате отдельно на каждом из четырех срезовых тестовых датасетов, то есть эффективность подхода была проверена на 4 клиентах независимо. Результаты представлены на графиках на рисунке 18. Из графиков можно сделать вывод о существовании сценариев, в которых даже при текущей реализации возможен ощутимый прирост скорости генерации. Отдельно стоит отметить, что данный прирост может быть вызван как на срезах, которых не было в обучающей выборке (первый клиент, соответствующий столбец в каждой группе для которого отсутствует за даты ранее 9 марта), так и на срезах присутствующих в данных для обучения (второй клиент). Для первого клиента удалось достичь прироста acceptance rate на величину до +82% (или 0.842 токена за шаг), что конвертировалось в +36% токенов в секунду, а для второго клиента наибольший прирост acceptance rate составил +60% токенов (или 0.673 токена за шаг), что ускорило генерацию токенов на +24%.

Характерной особенностью биграммного метода обусловливания на клиента является то, что наибольший прирост достигается на срезах с более простыми и предсказуемыми ответами: особенностями первых двух срезов являются, соответственно, короткие ответы и требование соответствия ответа четкой структуре, а последние два среза требуют от модели, соответственно, развернутый ответ и придумывание материала на заданную тему.

Дополнительно был проведен эксперимент, в котором подход с априорным распределением сравнен с моделью, переобученной под одного клиента. Из четырех клиентов был выбран единственный, данные от которого присутствовали в обучающей выборке и для которого был получен ощутимый прирост. Результаты сравнения изображены на рисунке 19. Из графиков видно, что качество модели, переобученной под один срез, после резкой деградации, вызванной изменением в потоке запросов, начинает показывать себя существенно хуже и, в конечном итоге, уступает драфтовой модели, обученной на общем потоке, с адаптивным априорным распределением.

Это говорит о том, что предлагаемый подход позволяет системе адаптироваться к изменениям в распределении запросов от клиента даже если модель спекулятивного декодирования начинает деградировать. Данное качество наиболее полезно в промышленных сценариях,

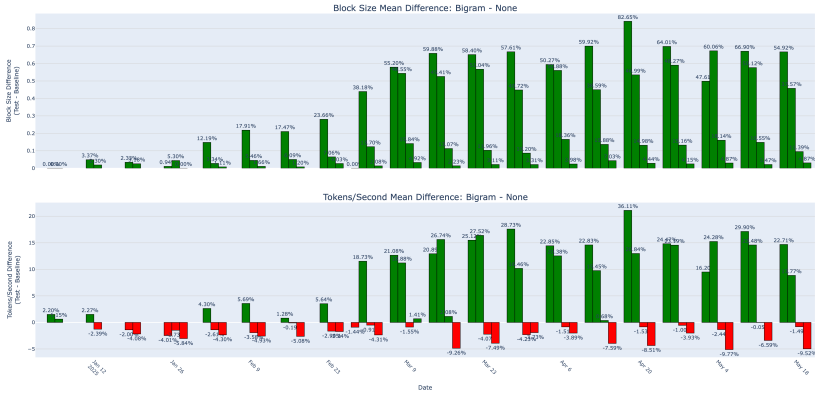

Рис. 18. Разница (абсолютная и относительная) метрик спекулятивного декодирования при использовании биграммного априорного распределения и без его использования (соответственно, test и baseline) на четырех срезовых тестовых датасетах. Первый срез характерен краткостью и предсказуемостью ответов (тема обращения в одном предложении), а второй срез характерен требованием ответа на запрос в четкой структуре (в формате JSON). Третий срез предполагал развернутый ответ на вопрос, а четвертый срез просил придумать материал под заданный сюжет.

в которых обновление любой нейросетевой модели является трудоемким и затратным по времени процессом, в то время как недостаточная пропускная способность системы с такой моделью способна пагубно сказаться на конкурентноспособности сервиса, предоставляющего доступ к данной системе.

§5. Заключение и будущая работа

В данной работе был приведён подробный обзор современных методик спекулятивного декодирования, а также приведена оценка с точки зрения применения в составе мультиарендных сервисов. Предложен метод клиентообусловленного спекулятивного декодирования (CCSD),

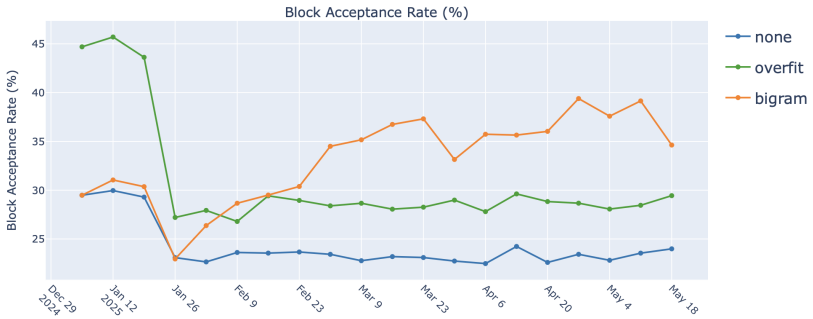

Рис. 19. График изменения acceptance rate спекулятивного декодирования в трех постановках. Синий график соответствует модели, обученной на всем потоке, зеленый – модели, обученный только на рассматриваемом потоке, а желтый – модели для всего потока со встроенным обусловливанием на клиентов при помощи биграммных статистик.

демонстрирующий перспективные результаты в контексте мультиарендных сервисов. Тем не менее ряд вопросов остаётся за рамками проведённого исследования, например, технической реализации предлагаемого подхода, не оказывающего существенного влияния на время работы драфтовой модели, а также проведение экспериментов в рамках настоящих мультиарендных сервисов с высокой нагрузкой, – их проработка составит основу дальнейшей работы.

В перспективе предполагается усовершенствовать механизмы персонализации, выйдя за рамки униграммных и биграммных априорных распределений, и добавить возможность использования n-грамм или иерархических априоров, учитывающих контекст разной глубины. Кроме того, представляет интерес интеграция CCSD с другими методами спекулятивного декодирования, такими как EAGLE-3, для получения наилучших результатов в ускорении генерации ответов системой с большой языковой моделью.

Помимо этого, для обеспечения дополнительного ускорения генерации целесообразно расширить словарь токенов за счёт применения

алгоритма byte-pair encoding (BPE). Добавление в словарь новых токенов, представляющих собой объединения элементов исходного словаря, способно повысить эффективность спекулятивного декодирования, поскольку принятие одного токена из расширенного словаря будет эквивалентно принятию двух и более токенов из первоначального набора за один шаг спекуляции. Это открывает возможности для более компактного представления последовательностей и потенциально может привести к значительному росту пропускной способности системы.

Существенный резерв повышения производительности кроется в инженерной оптимизации процесса декодирования. Перспективным выглядит совмещение двух режимов работы: базового прогона с исходным словарём и спекулятивного прогона с расширенным набором токенов. При этом для адаптации к различным срезам данных предлагается задействовать дообученные LoRA-адаптеры. Такой гибридный подход может позволить:

- Сократить время генерации благодаря параллельной обработке и предварительному вычислению кандидатов;
- Сохранить качество генерации, обеспечиваемое целевой моделью, за счёт верификации на уровне исходной лексики;
- Гибко адаптироваться к разнородным клиентским запросам, динамически переключаясь между режимами в зависимости от контекста.

Список литературы

1. Y. Cheng, A. Zhang, X. Zhang, C. Wang, and Y. Wang, *Recurrent Drafter for Fast Speculative Decoding in Large Language Models*, ArXiv preprint arXiv:2403.09919 (2024).
2. A. Graves, *Sequence Transduction with Recurrent Neural Networks*, ArXiv preprint arXiv:1211.3711 (2012).
3. E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, *LoRA: Low-Rank Adaptation of Large Language Models*, ArXiv preprint arXiv:2106.09685 (2021).
4. Y. Leviathan, M. Kalman, and Y. Matias, *Fast Inference from Transformers via Speculative Decoding*, ArXiv preprint arXiv:2211.17192 (2023).
5. Y. Li, F. Wei, C. Zhang, and H. Zhang, *EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees*, ArXiv preprint arXiv:2406.16858 (2024).
6. Y. Li, F. Wei, C. Zhang, and H. Zhang, *EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test*, ArXiv preprint arXiv:2503.01840 (2025).
7. X. Liu, L. Hu, P. Bailis, A. Cheung, Z. Deng, I. Stoica, and H. Zhang, *Online Speculative Decoding*, ArXiv preprint arXiv:2310.07177 (2024).

8. C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*, ArXiv preprint arXiv:1910.10683 (2023).
9. L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, *et al.*, *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*, ArXiv preprint arXiv:2306.05685 (2023).

Angeni G. E., Ershov V. A. Client-conditioned speculative decoding for large language models in multi-tenant services.

This work focuses on optimizing the application of large language models (LLMs) in multi-tenant services. It contains a comprehensive review of modern speculative decoding methods, and proposes Client-Conditioned Speculative Decoding (CCSD) to introduce prediction personalization without affecting the quality of generated responses. CCSD is integrated on top of existing speculative decoding techniques and utilizes client-specific n-gram statistics, updated online, which are used to adapt the distribution of candidate tokens produced by the draft model during speculative sampling. Experiments show that on client request streams with predictable responses, using bigram statistics boosts the number of tokens generated per second by up to +36% and the acceptance rate by up to +82%. Moreover, the ability of CCSD to adapt to changes in client query distribution during operation allows maintaining a higher performance gain compared to fine-tuning the draft model for a specific client.

Лаборатория искусственного интеллекта,
математики и компьютерных наук им. А. А. Маркова,
199034, Университетская наб. 9,
Санкт-Петербург, Россия

E-mail: geaned@yandex-team.ru, noxoomo@yandex-team.ru

Поступило 12 декабря 2025 г.