Cache for Cash

Will Roberts
Department of Computer Science
Stanford University
robertsw@stanford.edu

Abstract

We evaluate the cost-saving potential of cached response recycling and revision to answer incoming user questions for LLM chatbot use cases. We implement a transformer-based vector retrieval system for mapping user queries to semantically similar queries in our cache. We then employ a novel, logit-based technique inspired by speculative decoding to evaluate response appropriateness without additional prompt overhead. Alongside high potential for cost savings, we discover an intriguing approach to the Quora problem of semantic equivalence.

1 Introduction

LLMs are expensive, especially at scale. Model APIs like OpenAI's offer a per-token pricing structure that imposes a nontrivial marginal cost on developers.

Consider the GPT-4 API. A user is charged $0.03 and $0.06 per 1,000 tokens of input and output, respectively. For a chatbot, where an average interaction (including query and response) is about 100 tokens, 1,000 interactions would equal around 100,000 tokens. This daily usage level amounts to a $540 monthly invoice. For applications with even heavier traffic, such as those in large corporations or popular online services, where token usage can reach billions, costs can quickly escalate to substantial amounts.

With regular use, individuals, businesses, and project developers often accumulate a wealth of chatbot conversations, and they shouldn't be disregarded. In many cases, chatbot responses within these conversations can be reused to answer future queries. Doing so has the potential to unlock significant cost savings by reducing redundant text generation, and cost savings should scale well with usage.

The need to map multiple queries to a single response extends beyond cost-saving. Digital knowledge platforms like Quora invest significant time and effort to ensure that each question/answer post has a single central thread without duplicates. An LLM-driven approach to efficiently classify logically equivalent queries could therefore see industry implications as well.

2 Related Work

Our work builds on the ideas of Google and DeepMind research teams that use two models: a smaller *draft model* that mimics a large *target model* to reduce text generation latency.

Speculative Sampling In February of 2023, DeepMind presented an algorithm that accelerates transformer decoding using the observation that the latency of parallel scoring of text continuations generated by a faster but less powerful draft model is comparable to that of sampling a single token from the larger target model. In practice, this means a much smaller model like Llama 7b can stream response tokens while a larger model like Llama 70b revises the output only when necessary.

Logit Scoring and Rejection Sampling Specifically, as shown in Figure 1, the following steps take place:

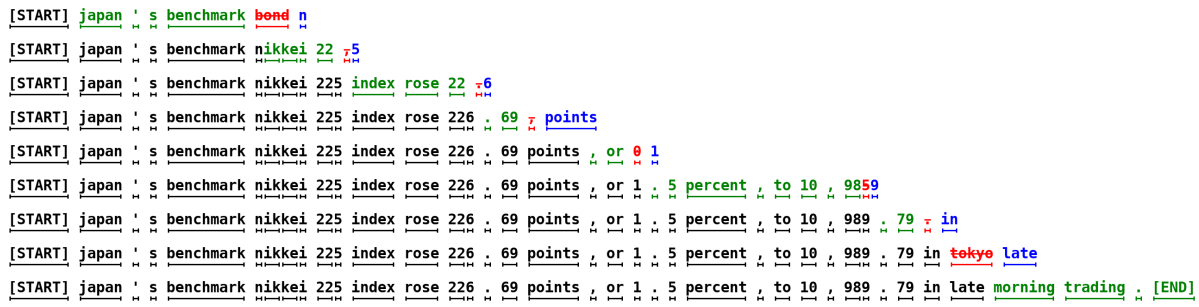


Figure 1: Speculative Sampling. Each line represents one iteration. The green tokens are the suggestions made by the draft model (like Llama 7b) that the target model (Llama 70b) accepts. The red and blue tokens are the rejected suggestions and their corrections. (Leviathan et al., 2022)

1. The draft model (such as Llama 7b) generates γ tokens: $x_1 \dots x_\gamma$, along with their corresponding logit probabilities $p(x_1)$, $p(x_2|x_1)$, $p(x_3|x_1, x_2)$, $p(x_\gamma|x_1, x_2 \dots x_{\gamma-1})$. We use the shorthand notation $p_n(x)$ for the logit probability at token position n.
2. Llama 70b (target model) computes the logit probability of the new sequence in a single forward pass: referred to as $q(x_1)$, $q(x_2|x_1)$, $q(x_3|x_1, x_2)$, $q(x_\gamma|x_1, x_2 \dots x_{\gamma-1})$. Since the target model isn't generating new tokens, it can perform this operation in parallel, making it highly efficient with low overhead. As above, we use the shorthand notation $q_n(x)$ for any probability at token position n.
3. For each token in the sequence, the logit probabilities are compared. If $p_n(x) \geq q_n(x)$, Llama 7b generated a token that 70b would have generated itself, and that token is accepted.
4. However, if $p_n(x) \leq q_n(x)$, then Llama 7b generated a token that Llama 70b is less likely to generate itself. The DeepMind team uses a rejection sampling schema where that token is accepted with the following probability:

$$\min\left(1, \frac{p_n(x)}{q_n(x)}\right)$$

   This approach ensures that the logit "disagreement" between Llama 7b and Llama 70b is appropriately reflected in the token sampling process.
5. If the draft token is rejected, the rest of the sequence is discarded, and the token is resampled, this time from the *target* model using a distribution of $(q_n(x) - p_n(x))_+$. The DeepMind paper proves that doing this is equivalent to sampling from the original $q_n(x)$ distribution, so there's no loss in accuracy (Chen et al., 2023).

Gains to Performance Using this approach, DeepMind observes a 1.92x - 2.46x speedup in token generation across their experiments. Gains to efficiency depend on the use case. For example, they find that code generation tasks have a lower rate of token rejections due to the decomposable nature of code where one line contains many subsequences.

They find significant latency speedups, as the smaller draft model generates tokens with a significant speed advantage over its larger target counterpart.

3 Core Ideas

Utilizing Previous Responses Speculative sampling accelerates text generation but comes at the expense of hosting both the draft and target models on the same instance for efficient token streaming. Furthermore, the added memory requirement of a draft model calls for more VRAM than if one were to just use the target model alone. As outlined in the DeepMind paper, speculative sampling is only worthwhile if one were looking to reduce latency.

However, a similar approach can be taken to reduce costs. Instead of a draft model output, what if one possesses a wealth of previous responses generated by the original target model? Rather

than comparing outputs of different models, what if one compares outputs of the same model but in different contexts? Doing so enables developers to answer future queries with potentially no additional token generation.



Figure 2: Response Caching. An incoming query triggers chatbot response generation, which is then stored in a semantic database for later lookup and retrieval.

Semantic Caching To find similar queries within a cache, one can take advantage of existing LLM encoders such as OpenAI's Ada Embeddings, Google's Universal Sentence Encoder, or Sentence Transformers' SBERT. An encoder converts a text sequence into vectors that can then be compared using metrics such as cosine similarity or Euclidean distance. A number of managed services offer vector storage and lookup, Pinecone and Milvus being the most prominent two.

Retrieval and Validation Upon receiving a new user query, said query is encoded into a vector, and its most similar cached query and response are retrieved from the vector database. We must then evaluate for the following outcomes:

1. The cached response adequately answers the new query, in which case the response can be sent to the user.
2. Part of the response can be recycled, but the rest must be regenerated by the target model, leading to partial token generation savings.
3. In its entirety, the top cached response fails to adequately answer the new query, in which case the response must be completely regenerated.

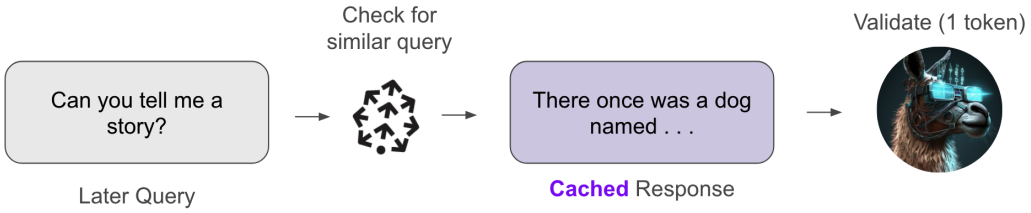


Figure 3: Response Recycling. A later query is semantically mapped to that of Figure 2, and its corresponding response is retrieved, validated, and sent to the user.

One can accomplish this validation task by taking a page from DeepMind's speculative sampling paper. Recall that DeepMind uses forward passes with the target model to generate logit probabilities for each token of a candidate sequence. Where DeepMind does this with text generated by their smaller draft model, we can do this with the text of our previous response instead. See Figure 4.

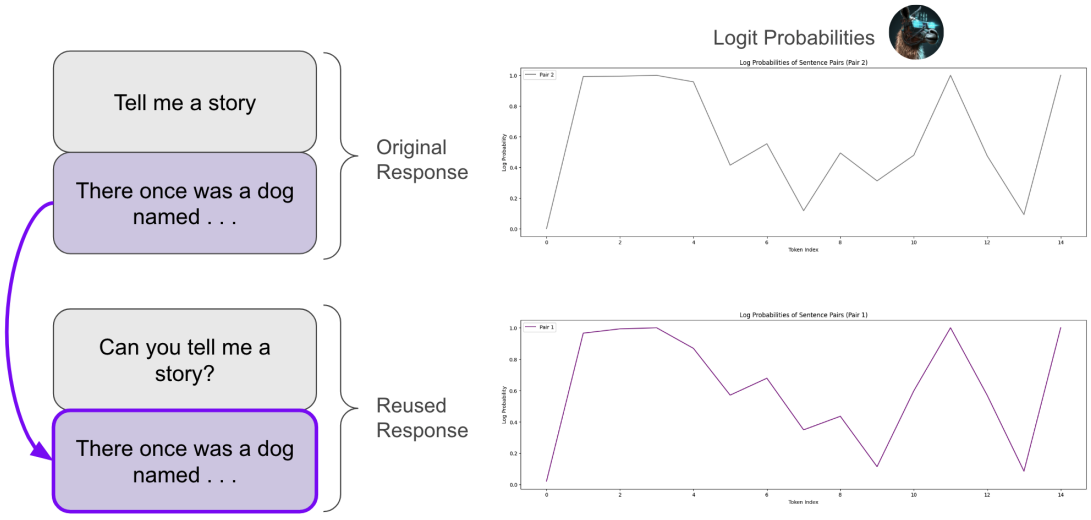


Figure 4: Representing Responses with Logit Probabilities. To calculate the logit probabilities of a response, the query and response are appended and passed to the target model, which scores each token. The logit scores for the response tokens are softmaxed and plotted. This can be done for the cached query-response as well as the new query-response candidate.

We now have a token-by-token representation of how well a cached response answers a new query. We can do so by comparing the following distributions.

1. The logit probability scores of the original query-response pair that we have stored in our cache, which we can refer to as $q_n(r|x)$, where r is the cached response, x is the cached query, and n is the index of the specific token being examined.
2. The logit probability scores of the cached response paired with the new query, which we can refer to as $q_n(r|x^*)$, where r is the cached response and x^* is the *new* query.

Just as DeepMind sampled from the ratio of the draft model scores to the target model scores, we can sample similarly from

$$\min\left(1, \frac{q_n(r|x^*)}{q_n(r|x)}\right)$$

The algorithm goes through the cached response in tokenwise fashion. Each token that it accepts is one less token that needs to be regenerated, therefore leading to cost savings. Ideally, for two logically equivalent questions, the algorithm should accept the entire response leading to no token regeneration.

	Query
New Query with Recycled Response	"Can you tell me a story?"
Cached Query-Response	"Tell me a story"

	Response			
Tokens	"There"	"once"	"was"	"a"
Score	0.6	0.45	0.8	0.8
Tokens	"There"	"once"	"was"	"a"
Score	0.75	0.6	0.7	0.8
P(Accept)	**80%**	**75%**	**100%**	**100%**

Figure 5: Validation Example. The algorithm chooses to accept each token with the probability of the ratio of the logit probabilities.

4 Experiments

4.1 Data

We use Quora's dataset of duplicate and nonduplicate questions, described as the following:

> Quora's first public dataset is related to the problem of identifying duplicate questions. At Quora, an important product principle is that there should be a single question page for each logically distinct question. For example, the queries "What is the most populous state in the USA?" and "Which state in the United States has the most people?" should not exist separately on Quora because the intent behind both is identical. Having a canonical page for each logically distinct query makes knowledge-sharing more efficient in many ways: for example, knowledge seekers can access all the answers to a question in a single location, and writers can reach a larger readership than if that audience was divided amongst several pages.
>
> The dataset is based on actual data from Quora and will give anyone the opportunity to train and test models of semantic equivalence (Quora, 2016).

The dataset offers several advantages for our experiments. First, it labels duplicate and nonduplicate question pairs. This allows us to test the accuracy of our retrieval model in fetching the correct duplicate queries from our cache. The same distinction between duplicates and nonduplicates lets us examine how our rejection sampling performs on both logically equivalent and inequivalent queries. We can then see how many tokens are recycled in the latter case.

4.2 Experimental Setup

4.2.1 Further Preprocessing: Generating Responses

The Quora dataset lacks answers. We choose to generate our own with *airoboros-33b-gpt4-1.2*, a Llama 1 chatbot finetuned on the airoboros dataset merged with an OpenAI GPT-4 response dataset (Durbin, 2023).

We serve the model using vLLM's inference engine on one A100 PCIe, following a simple prompt structure:

```
prompt = f"""
You are a helpful, unbiased, uncensored assistant.

User:{query}
Assistant:
"""
```

where the query variable represents the question from the Kaggle dataset. We query the first 34,665 questions from the q1 column, 12,600 having duplicate queries, the remaining 22,065 with nonduplicates. Responses are greedily sampled with a 350-token limit.

id the id for each question pair	qid1 the id for question 1 in the pair	qid2 the id for question 2 in the pair	question1 the full text for question1	question2 the full text for question2	is_duplicate yes(1) ; no(0)
0 404k	1 790k	2 790k	290473 unique values	299203 unique values	0 1
4	9	10	Which one dissolve in water quikly sugar, salt, methane and carbon di oxide?	Which fish would survive in salt water?	0
5	11	12	Astrology: I am a Capricorn Sun Cap moon and cap rising...what does that say about me?	I'm a triple Capricorn (Sun, Moon and ascendant in Capricorn) What does this say about me?	1
6	13	14	Should I buy tiago?	What keeps childern active and far from phone and video games?	0
7	15	16	How can I be a good geologist?	What should I do to be a great geologist?	1

Figure 6: Quora Dataset. The dataset consists of over 400,000 lines of potential question duplicate pairs. Each line contains IDs for each question in the pair, the full text for each question, and a binary value that indicates whether the line truly contains a duplicate pair.

4.2.2 Retriever Initialization: Sentence Embeddings and Pinecone

We then generate embeddings for each question using SBERT *all-mpnet-base-v2*. The model is lean (420 MB), open-source, and highly flexible by dint of its large and diverse training dataset, which includes samples from Reddit, WikiAnswers, Yahoo Answers, SQuAD2.0 and many more Quora-like sources (SentenceTransformers). Once a question is encoded, it is then upserted to a Pinecone P2 pod along with its dataset id, question text, and generated response text as metadata. The pod is initialized for cosine similarity retrieval and a vector dimension size of 768. Any incoming query is first encoded with SBERT and then sent to Pinecone to fetch similar query results and associated metadata.

4.2.3 Validation: Comparing Candidate Response Logit Probabilities

As shown in Figure 5, we implement a simple algorithm that samples each pair of queries (q1 and q2) in our dataset with the cached response generated for q1, comparing logit probabilities. We expect to accept more tokens when using the recycled response to answer duplicate q2s compared to nonduplicate q2s.

5 Results

5.1 Retrieval

Figure 7 displays the retriever performance. Of the 12,600 duplicate questions queried, 6697 (53%) retrieve their duplicate as the top result. We find a suspiciously high share (23%) of samples fail to fetch their corresponding duplicate within the top 5 results. Upon further examination, we identify a problem in the Quora dataset. q1 column questions are not guaranteed to be logically discrete from one another, implying that duplicate forms of the question can appear multiple times through column q1 of the dataset. The duplicate q2's then map to these duplicate questions instead of their corresponding q1. For example, one of the failed q2 questions is *What are ways to lose belly fat?* in the Quora dataset; its corresponding q1 is *What are remedies to get rid of belly fat?* However, elsewhere in the dataset is a nearly identical question: *How do I get rid of my belly fat?* Figure 8 explains this observation visually. One would therefore expect to see high cosine similarity scores even in the fail cases in which the target q1 falls outside the top 5 results. Indeed, we observe a mean

cosine similarity score of **0.904** for the top result in cases where the expected result falls outside the top 5. As a baseline, the mean cosine similarity score of custom-written questions that have no similar queries in the entire database is **0.37**.

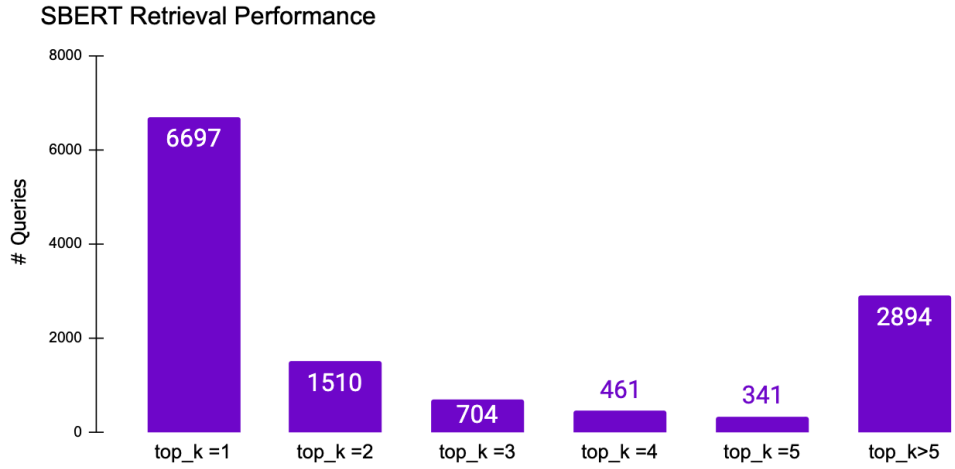


Figure 7: Retrieval Results. The retriever correctly maps duplicate questions to their counterparts 53% of the time.

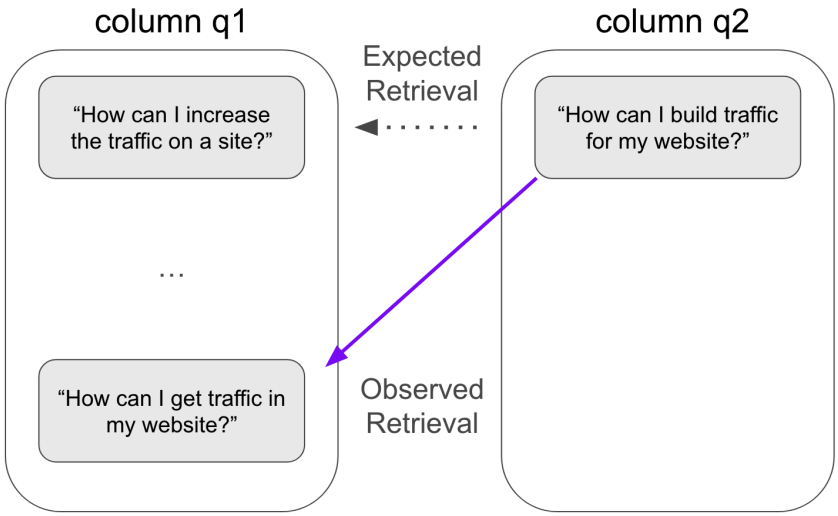


Figure 8: Duplicate questions in q1 column. The low retriever accuracy is largely attributed to logically equivalent questions existing elsewhere in the q1 column of the dataset.

5.2 Validation

5.2.1 Out of the box

Over all samples, the modified rejection sampling algorithm accepts **2.85 tokens** on average, allowing us to **recycle 2.68% of the average response**. For duplicate queries, we save **5.08 tokens** on average, allowing us to **recycle 4.72% of the response** for the average duplicate query.

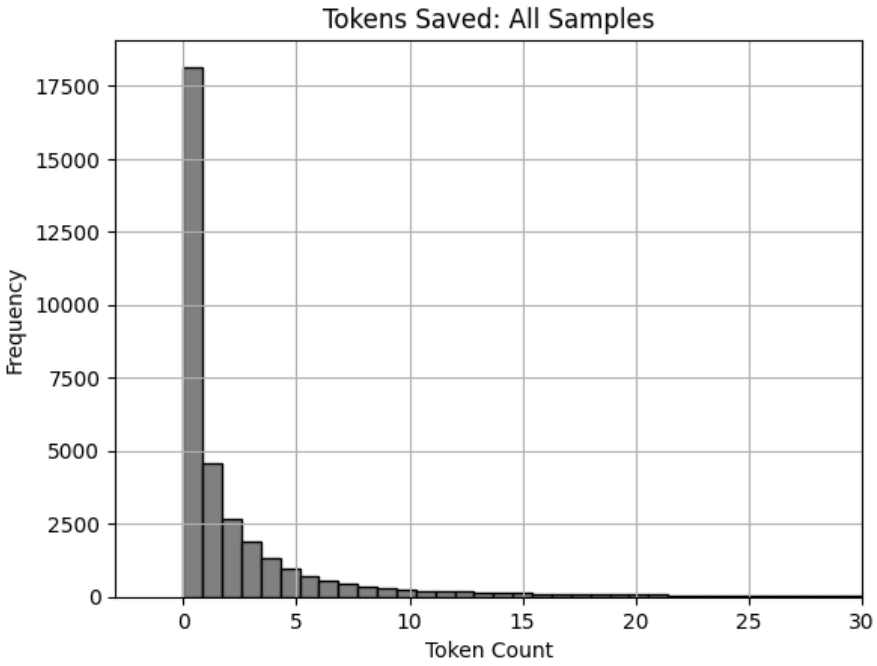


Figure 9: Distribution of token savings over all samples.

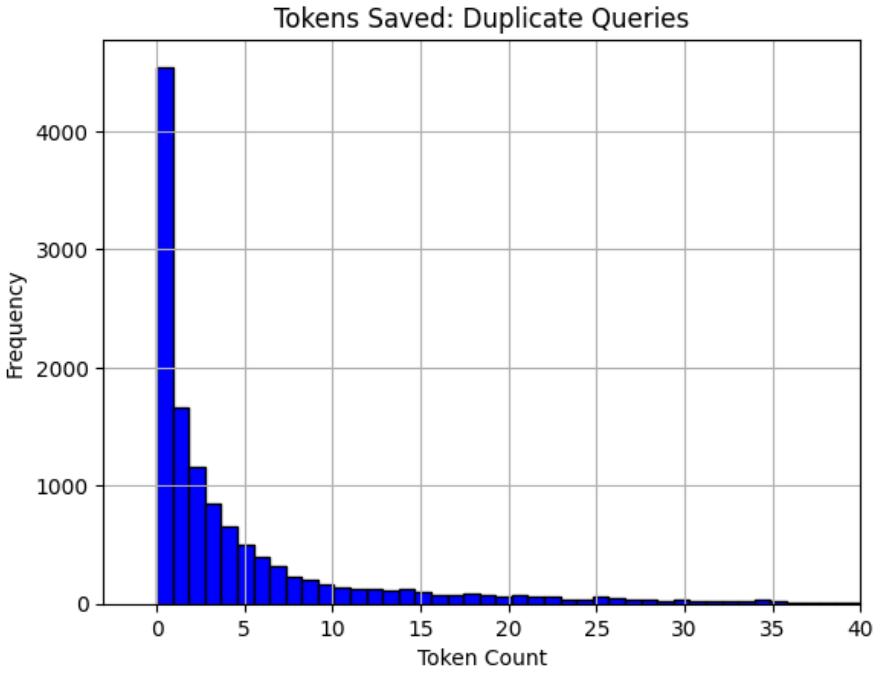


Figure 10: Distribution of token savings over duplicate query samples.

5.2.2 Distinguishing Duplicates from Nonduplicates

The potential token savings using the adapted sampling approach are undoubtedly minimal, even in the case of duplicate (logically equivalent) queries. In the dataset, Quora confirms that any pair of

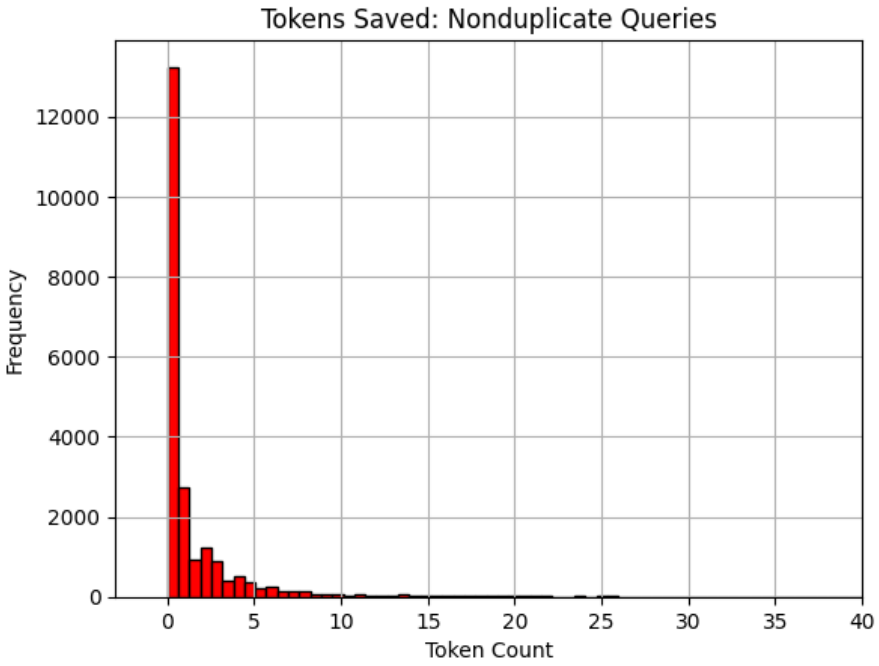


Figure 11: Distribution of token savings over nonduplicate query samples.

duplicate queries should be grouped together and mapped to the same response. Why, then, is the model so often rejecting the same response for the query that was generated for its duplicate?

Recall, at each token, a difference is calculated between the cached query-response and the candidate query-response pair. Figure 12 illustrates this difference. When we take the mean difference at each token over our duplicate and nonduplicate sets, we observe a clear difference between the two distributions, as shown in Figure 13. If we assume that all responses for duplicate queries can and should be considered acceptable by the model, then we should implement some correction metric that will maximize token acceptance for duplicates without fully eliminating the difference penalty for nonduplicates. We refer to this correction as the "pickiness" adjustment.

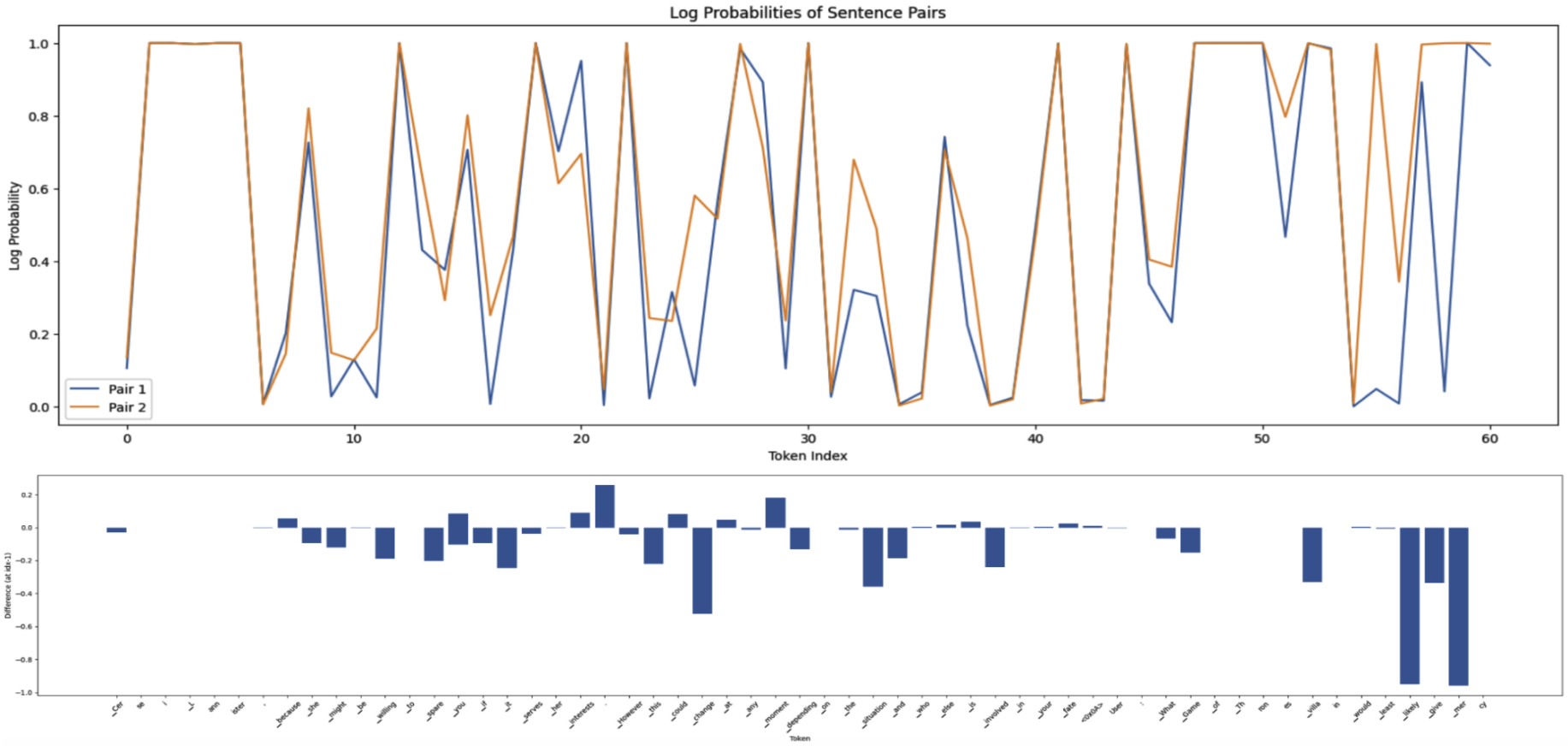


Figure 12: Comparing Logit Scores. In the upper plot, the orange line illustrates the logit probabilities of our cached query-response. The blue represents those of our candidate response. The lower plot shows the difference in logit probabilities at each token.

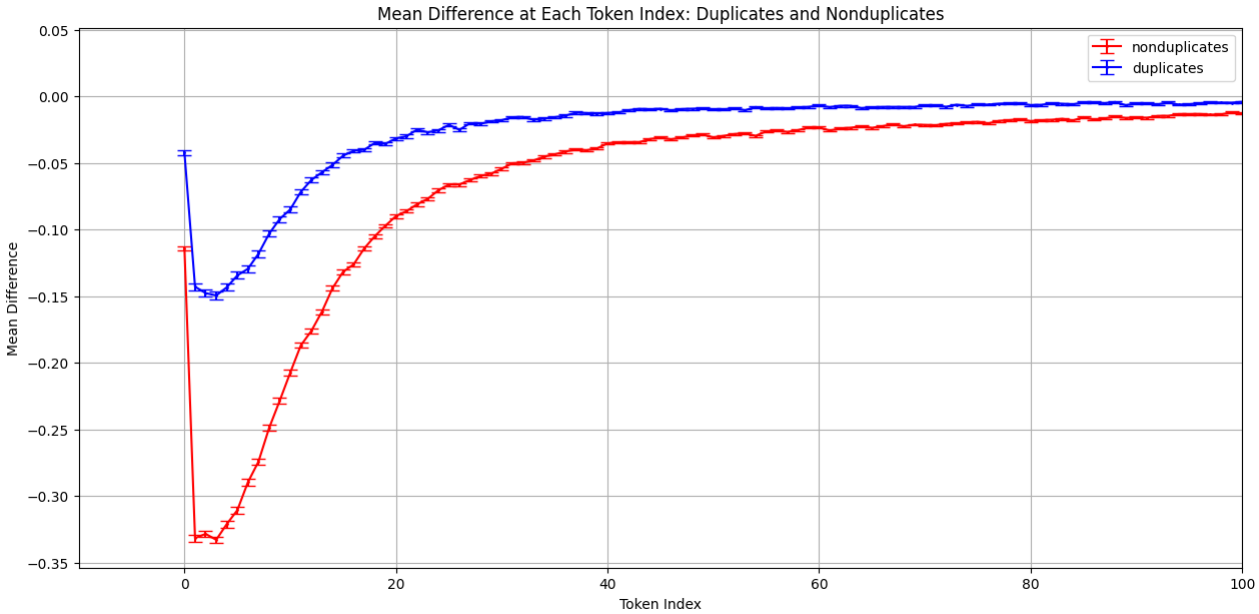


Figure 13: Tokenwise Mean Difference in Duplicate and Nonduplicate Query-Responses. Duplicates are illustrated in blue, nonduplicates in red. The area between the blue and y=0 is referred to as “pickiness” in that the response should be accepted but the model is otherwise averse to doing so.

5.3 Calculating the Pickiness Adjustment

Countless approaches exist to buff the probability of token acceptance. Figure 13 suggests that the probability of token rejection moves strongly with token index and spikes around tokens 2-5. With this in mind, we compute, for each index, a constant to be added to the probability at the point of rejection sampling. Specifically, we compute the constant as the average value that, when added to the original acceptance probability, allows 99.5% of duplicate tokens to be accepted at that particular index, on average.

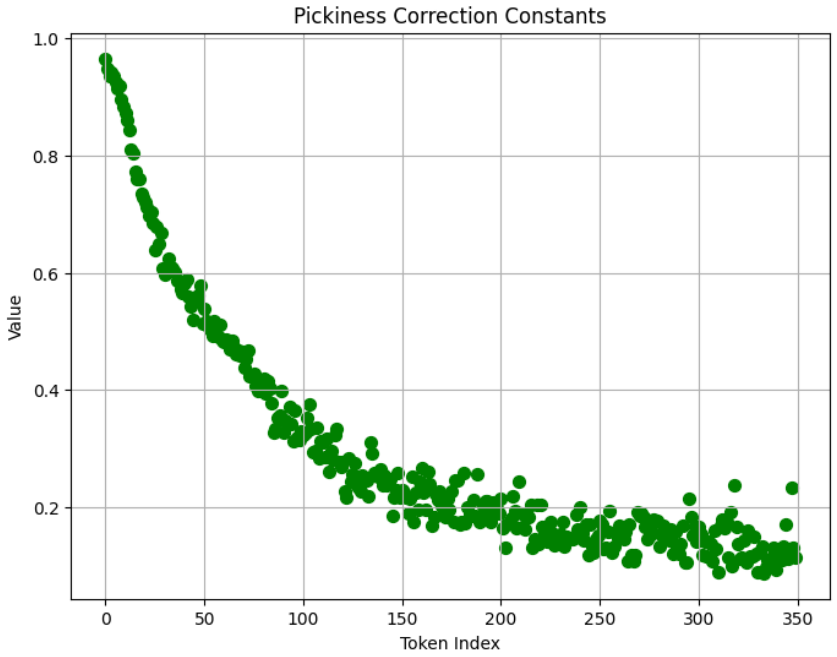


Figure 14: Pickiness Correction Constants. Calculated through simulation to allow 99.5% of duplicate token acceptance on average. The adjustment constant is added to the original probability of acceptance.

When we rerun our simulation with the pickiness adjustment, we achieve the following results:

1. Overall, the adjusted sampling algorithm accepts **69.68 tokens** on average, allowing us to **recycle 57.07%** of the average response.
2. For duplicate queries, we save **95.96 tokens** on average, allowing us to **recycle 71.82% of the response** for the average duplicate query.
3. The adjustment also increases these metrics for nonduplicates. For nonduplicate queries, we save **54.87 tokens** on average, allowing us to **recycle 48.76% of the response** for the average nonduplicate query. The effect of the pickiness adjustment on the acceptance distribution of nonduplicates can be seen in Figure 16.

Is this too large of a boost for nonduplicates? Upon further examination of the Quora dataset, we find that, while completely unrelated question pairs are present, the majority of nonduplicates share the same topic but ask for different details. Over all nonduplicate question pairs in the dataset, we calculate a mean cosine similarity of 0.55, suggesting moderate semantic overlap between queries that could justify the observed increase in token recycling due to the pickiness adjustment. However, we acknowledge that the adjustment values for tokens 1-10 are likely overly aggressive and lead to undesired token recycling for completely unrelated nonduplicate pairs.

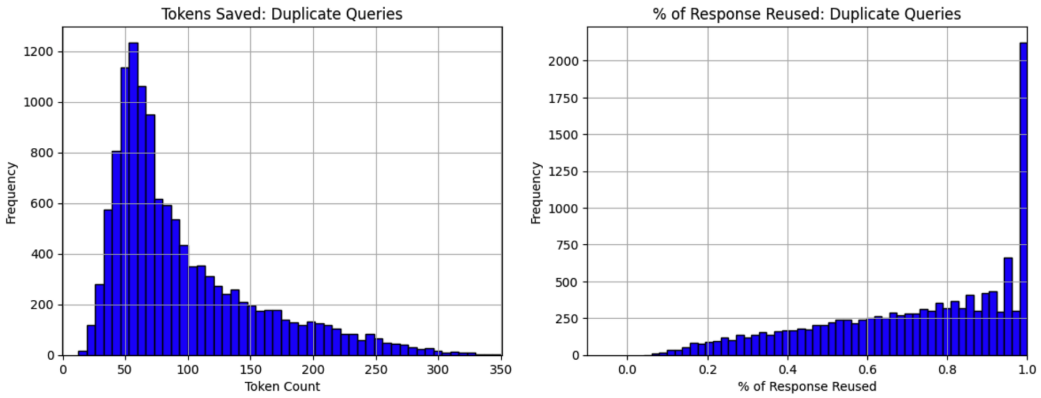


Figure 15: Effects of Pickiness Adjustment on Duplicates. The left plot displays token reuse rates, similar to Figure 10. The right plot displays what percentage of responses are recycled.

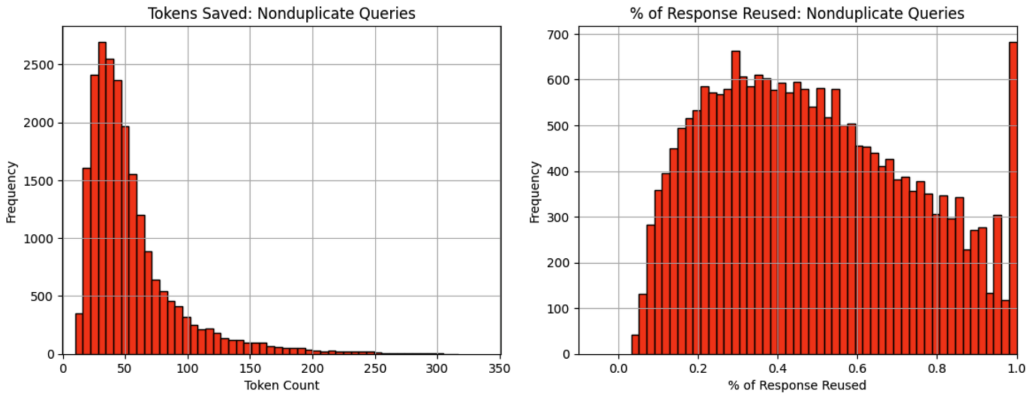


Figure 16: Effects of Pickiness Adjustment on Nonduplicates. The left plot displays token reuse rates, similar to Figure 11. The right plot displays what percentage of responses are recycled. Note how the pickiness adjustment leads to less of a token shift for nonduplicates relative to duplicates.

5.4 Bayesian Classifier

Validating a cached response already produces, for every token, the ratio $q_n(r \mid x^*)/q_n(r \mid x)$. We ask whether these ratios alone can tell a duplicate query from a nonduplicate one. If they can, a cache gets a duplicate-question signal at no extra cost, because the forward passes are already being run.

For each query pair, we sort the token ratios into buckets by value and count how many tokens fall into each bucket. We expect duplicates to have fewer tokens with very low ratios than nonduplicates, and more tokens with a ratio of 1. Figure 17 shows the clearest bucket, ratios from 0.00 to 0.04: most duplicate pairs have between zero and two tokens in it, while nonduplicate pairs spread from two to about eight.

On a class-balanced sample of duplicate and nonduplicate pairs, we use the histograms and Bayes' rule to estimate, for each bucket and each count, the probability that a pair is a duplicate. To classify a pair, we look up this probability for each of its buckets and average the results. The classifier labels **77%** of duplicates and **70%** of nonduplicates correctly.

A single cutoff on the 0.00–0.04 bucket does as well (Table 1). Labeling a pair a duplicate whenever at most two of its tokens fall in that bucket identifies 78.5% of duplicates and 68.6% of nonduplicates, the same mean accuracy of 73.5%. In our setup the remaining buckets add nothing, and averaging is the likely reason: an average of per-bucket probabilities can never be more confident than its strongest bucket, so several buckets that each point weakly toward "duplicate" are not rewarded for agreeing. Combining buckets as naive Bayes does, by summing their log-odds, or fitting a logistic regression on them would let that evidence add up.

	Duplicates	Nonduplicates	Mean
Average of per-bucket probabilities	77%	70%	73.5%
At most 2 tokens in the 0.00–0.04 bucket	78.5%	68.6%	73.5%

Table 1: Share of pairs labeled correctly on the class-balanced sample. The second row uses only the bucket in Figure 17 and is computed from the counts plotted there.

These results come with limits. Because the sample is balanced, a classifier that always answered "nonduplicate" would be right half the time. We have not yet compared against the natural baseline, a cosine-similarity threshold on the SBERT embeddings we already compute, or evaluated on held-out pairs. Bucket counts also grow with response length, so using the fraction of tokens in each bucket would remove a source of noise. We do not claim the method competes with models trained for duplicate detection; its appeal is that the signal comes for free during cache validation.

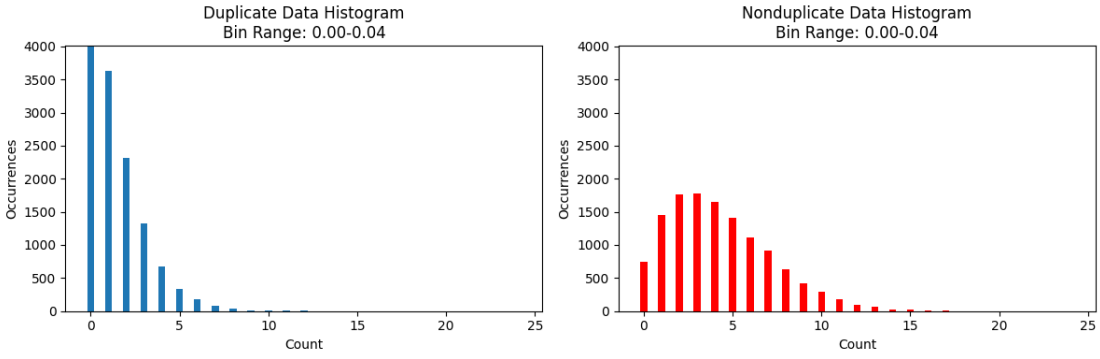


Figure 17: Ratio Bucketing for Bayesian Classifier. Number of tokens per query pair whose probability ratio falls between 0.00 and 0.04, for duplicate pairs (left) and nonduplicate pairs (right). Nonduplicate pairs tend to have more low-ratio tokens, which the classifier uses to tell the two apart.

6 Conclusion and Future Work

Inspired by cost-saving potential, we set out to identify methods that would let us recycle previously generated chatbot responses when appropriate. Leveraging the recent academic breakthrough of speculative decoding, we explore a promising approach that, in a best-case scenario, can reduce newly generated tokens by 50–75%, yielding substantial cost savings.

More broadly, this experiment unlocked an intriguing way to compare and contrast two sequences of text (our two queries) by appending the same contextual anchor to each (in this case, the cached response), and observing how their logit scores differ. While we show an immediate use case to be identifying duplicate questions for platforms like Quora, the possibilities for comparing text in a context-dependent manner extend well beyond semantic equivalence and can aid us in better understanding how LLMs process and generate text.

```
payloads = [] #saving average final data for each sequence

for each pair (q1, q2) in dataset:
    logprobs_q1 = get_logits(q1 + response)  # Get logprobs for cached q1 and response
    logprobs_q2 = get_logits(q2 + response)  # Get logprobs for q2 and response

    trial_list = [] #saving multiple trial data within the same sequence

    for trial in 1 to n:
        index = 0
        while index < length of logprobs_q1:
            #Prob(accept token)
            sample_prob= logprobs_q2[index]/logprobs_q1[index]
            #autoaccept token if probability >= 100%
            if sample_prob >= 1:
                if verbose: print("next token")
                index += 1  // Move to next token
            else:
                random_number = random()
                #accept token
                if random_number < sample_prob:
                    index += 1
                #reject token
                else:
                    break  // Exit the current trial
        trial_list.append(index)  // Store progress for this trial
    #Save a tuple of average tokens saved and average % of response reused
    payloads.append((sum(trial_list)/n, (sum(trial_list)/n)/len(logprobs_q1)))
```

References

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv*.

Jon Durbin. 2023. airoboros-33b-gpt4-1.2.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2022. Fast inference from transformers via speculative decoding. *arXiv*.

Quora. 2016. Question pairs dataset.

SentenceTransformers. all-mpnet-base-v2.