Large Language Models in Education

Rupasinghe T. T. V. N.
Dept. of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka.
e17297@eng.pdn.ac.lk

Kalpana M. W. V.
Dept. of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka.
e17148@eng.pdn.ac.lk

Manohora H. T.
Dept. of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka.
e17206@eng.pdn.ac.lk

Dr. Damayanthi Herath
Dept. of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka.
damayanthiherath@eng.pdn.ac.lk

Prof. Roshan G. Ragel
Dept. of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka.
roshanr@eng.pdn.ac.lk

Dr. Isuru Nawinne
Dept. of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka.
isurunawinne@eng.pdn.ac.lk

Dr. Shamane Siriwardhana
Dept. of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka.
gshasiri@gmail.com

***Abstract*—As artificial intelligence (AI) rapidly advances, particularly in generative AI models and large language models (LLMs), the educational landscape stands on the brink of transformation. Generative AI models, driven by extensive training, exhibit exceptional abilities in creating human-like content across various modalities. Large language models, specifically designed for understanding and generating human-like language, present unprecedented opportunities in education. Intelligent tutoring systems, chatbots, virtual assistants, and content generation platforms are integrating LLMs to revolutionize learning experiences, automate grading, and produce tailored educational content. Despite their promise, challenges such as inappropriate responses, privacy concerns, and high costs need addressing. In order to lessen the high-cost connected with LLM platforms, this paper presents a novel solution: cost effective intelligent tutor. By implementing a cache system and local context for a specific course, computer architecture, this prototype showcases the potential for widespread application across diverse educational modules, paving the way for more accessible, personalized, and efficient learning environments. Furthermore, the solution has demonstrated a remarkable 70% reduction in cost compared to the current systems, making it a cost-effective intelligent tutor for enhanced learning experiences.**

I. Introduction

As the rapidly developing discipline of computer science, artificial intelligence seeks to develop intelligent agents that can carry out tasks that generally require human intelligence [1], [2]. With that purpose, artificial intelligence has advanced significantly in the last few years, particularly in generative AI and large language models [3], [4]. These cutting-edge models have proven to be exceptionally capable of reading, writing, and producing content that is human-like, opening up new horizons in creativity and invention [5], [6], [14]. The purpose of generative AI models is to produce fresh, unique data using the knowledge and patterns acquired during training. These models can generate new instances of text, images, and other content not included in the training set. Large language models specifically refer to models focused on understanding and generating human-like language, often achieved through extensive pre-training on large datasets [9], [10].

These AI models provide educational prospects where technology integration is becoming more common. The most common system of LLMs in the educational field is the Intelligent tutoring system [7], [8]. This has been sought after in education for a long time to improve and customize students' learning experiences with tailored support, automated grading, and even generating educational content [15]. The effectiveness of generative AI paired with the current trends in educational technology use offers a potent synergy that has the potential to completely change the educational environment. Chatbots and virtual assistants, content generation platforms, code generation platforms, translation services etc., are the current platforms that use the LLMs in the educational field [16]. Chatbots and virtual assistants, equipped with Large Language Models (LLMs), enhance user interactions on educational platforms, answering queries and facilitating a more natural conversation. Content generation platforms leverage LLMs to assist users in creating high-quality educational content, while code generation platforms streamline programming tasks by generating code snippets based on natural language descriptions. In language education, translation services employing LLMs contribute to breaking down language barriers making educational content accessible to a global audience [17], [19]. Integrating LLMs across these platforms reflects a transformative shift in the educational landscape, fostering efficiency, personalization, and improved accessibility for learners and educators alike.

However, the large language models address many challenges [11], [12], [23] in these educational platforms [10], [18]. Some challenges include inappropriate responses, privacy concerns, high costs, and continuous improvement. A significant concern identified is the high cost of accessing Large Language Model APIs [13]. The price structure of LLM APIs is based on the usage volume measured in terms of API calls or tokens processed [13], [23], [36]. The more API calls or tokens used, the higher the associated cost. When there are millions of API calls given by multiple users, the resulting cost can be notably high [21], [22]. That is why it is called a major issue in LLM platforms. To address this challenge, a solution: cost effective intelligent tutor has been developed in our work. The proposed cost-effective intelligent tutor is a prototype that relies on a cache implementation with local context to achieve cost reduction. The methodology is implemented for computer architecture course and this prototype can be adopted for other courses.

II. Literature

There is a rapidly growing number of LLMs that users can query for a fee. We reviewed the cost associated with querying popular LLM APIs like GPT-4, ChatGPT, and J1-Jumbo [28] and found that these models have heterogeneous pricing structures, with fees that can digger by two orders of magnitude. Notably, the cost of utilizing LLMs for processing extensive collections of queries and text emerges as a potential financial concern. In response, three distinct strategies are analyzed that users can apply to mitigate the inference cost linked to utilising LLMs [13].

- Cost Reduction Methods
  1) Prompt adaptation
     a) Prompt Selection
     b) Query Concatenation
  2) LLM approximation
     a) Completion cache
     b) Model fine-tuning
  3) LLM cascade
- Cost Measurement when accessing LLM APIs

A. Prompt Adaptation

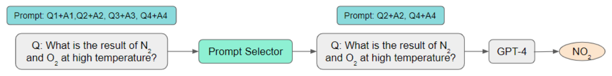

Prompt: Q1+A1,Q2+A2, Q3+A3, Q4+A4
Q: What is the result of N_2 and O_2 at high temperature?
Prompt Selector
Prompt: Q2+A2, Q4+A4
Q: What is the result of N_2 and O_2 at high temperature?
GPT-4
NO_2

Fig. 1: Prompt Selection

In exploring ways to make LLM APIs more cost-effective, one approach is adapting the size of the prompt. Fig 1 shows the data flow diagram of the prompt adaptation. The cost of a query to an LLM increases as the prompt size grows. So, a logical strategy is prompt adaptation, aiming to decrease the prompt size. Prompt selection is a practical example of this adaptation, where instead of having a prompt with many task-performing examples, we keep a smaller subset. This results in a more compact prompt and, consequently, a lower cost.

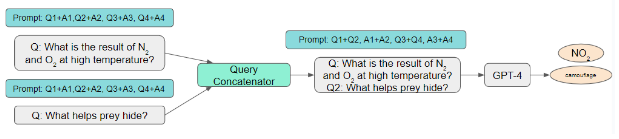

Prompt: Q1+A1,Q2+A2, Q3+A3, Q4+A4
Q: What is the result of N_2 and O_2 at high temperature?
Prompt: Q1+A1,Q2+A2, Q3+A3, Q4+A4
Q: What helps prey hide?
Query Concatenator
Prompt: Q1+Q2, A1+A2, Q3+Q4, A3+A4
Q: What is the result of N_2 and O_2 at high temperature? Q2: What helps prey hide?
GPT-4
NO_2
camouflage

Fig. 2: Query Concatenation

Another approach is query concatenation, which involves sending the prompt only once to the LLM API while having it address multiple queries. Fig. 2 shows the data flow diagram of query concatenation. This eliminates redundant prompt processing and involves combining several queries into one, explicitly instructing the LLM API to handle multiple queries in a single prompt. For instance, a prompt could include two queries and their respective answers, effectively managing both with a single prompt submission.

B. LLM Approximation

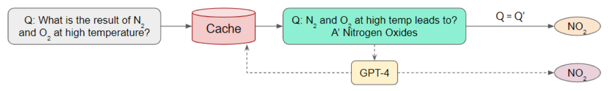

Q: What is the result of N_2 and O_2 at high temperature?
Cache
Q: N_2 and O_2 at high temp leads to? A' Nitrogen Oxides
Q = Q'
NO_2
GPT-4
NO_2

Fig. 3: Completion Cache

In making the use of expensive LLM APIs more budget-friendly, the idea of LLM approximation [13] comes into play. An example is the completion cache, which involves storing responses locally in a cache when sending a query to an LLM API. When a new query comes in, we first see if it has already been addressed in a similar manner. If so, we use the LLM API to get the response; if not, we get it from the cache. This completion cache is particularly cost-effective when dealing with frequent similar queries. For instance, in a search engine using an LLM API, if many users search for the same or similar keywords simultaneously, the completion cache allows us to answer all their queries by invoking the LLM only once [13].

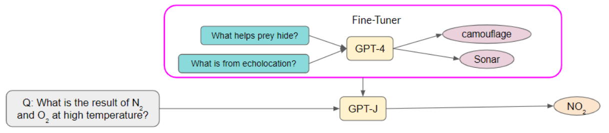

Fine-Tuner
What helps prey hide?
What is from echolocation?
GPT-4
camouflage
Sonar
Q: What is the result of N_2 and O_2 at high temperature?
GPT-J
NO_2

Fig. 4: Model fine-tuning

Fig. 4 shows another method for LLM approximation. It is called model fine-tuning, where we collect responses from an expensive LLM API, use them to fine-tune a smaller and more affordable AI model, and then employ this fine-tuned model for new queries. Besides cost savings, the fine-tuned model often brings latency improvements as a bonus, as it doesn't require lengthy prompts [13].

C. LLM Cascade

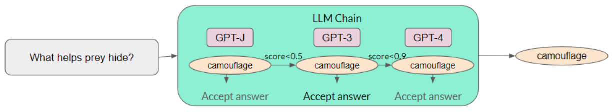


Fig. 5: LLM Cascade

Fig. 5 shows data flow diagram of LLM cascade approach. There's an intriguing avenue for data-adaptive LLM selection. Different LLM APIs have distinct strengths and weaknesses for handling various queries. Thus, choosing the right LLMs based on queries' specific nature can offer cost reduction and performance enhancements. The concept of LLM cascade involves sequentially sending a query to a list of LLM APIs. If the response from the first API is deemed reliable, it is returned, and there's no need to consult the subsequent APIs on the list. The remaining LLM APIs are only queried if the responses from the previous ones are considered unreliable. This approach significantly reduces query costs, especially when the initial APIs are relatively inexpensive and generate reliable responses. The two essential components of an LLM cascade are a generation scoring function and an LLM router, which is a model that learns the correctness of a generation based on the query and the generated answer. Learning the selected list and the threshold vectors involves modelling it as a constraint optimization problem [13].

In seeking ways to improve performance, one exciting approach is the joint selection of prompts and LLMs. This involves finding the smallest prompt and the most budget-friendly LLM that achieves satisfactory task performance for a given query. Another strategy involves exploring both existing LLM APIs and fine-tuned models. It's worth noting that combining different approaches does increase the computational costs for training. This opens the door to exploring trade-offs between query costs, task performance, and the computational resources required. It is a crucial consideration in optimizing the overall efficiency of these approaches [13].

D. Cost Measurement when accessing LLM APIs

We picked 12 Large Language Model (LLM) APIs offered by five major providers: OpenAI, AI21, CoHere, Textsynth, and ForeFrontAI. The details are in the table below, and the cost information was collected in March 2023. There are three components to the cost of utilizing these APIs. These are a fixed cost per request, input, which is correlated with the quantity of input tokens, and output, which is correlated with the quantity of generated tokens. Interestingly, there can be up to two orders of magnitude disparity in the costs of these LLMs. For instance, to process 10 million input tokens, GPT-J from Textsynth costs only $0.2, while OpenAI's GPT-4 requires $30. This illustrates the substantial cost variations among different LLMs, even for similar tasks [13]. Table I shows the summary of commercial LLM APIs.

API	Size/B	10M input tokens (USD)	10M output tokens (USD)	request (USD)
GPT-Curie	6.7	2	2	0
ChatGPT	NA	2	2	0
GPT-3	175	20	20	0
GPT-4	NA	30	60	0
J1-Large	7.5	0	30	0.0003
J1-Grande	17	0	80	0.0008
J1-Jumbo	178	0	250	0.005
Xlarge	52	10	10	0
QA	16	5.8	5.8	0
GPT-J	6	0.2	5	0
FAIRSEQ	13	0.6	15	0
GPT-Neox	20	1.4	35	0

TABLE I: Summary of commercial LLM APIs

E. FrugalGPT reduces Cost and Improves accuracy

FrugalGPT, a practical implementation of LLM cascade, stands out as a straightforward and adaptable approach. It learns the most effective combinations of LLMs for different queries, aiming to minimize costs while enhancing accuracy. In experiments, FrugalGPT demonstrated the ability to match the performance of the top individual LLM, such as GPT-4, but with a remarkable up to 98% reduction in costs [13].

FrugalGPT has selected 12 LLM APIs, and it has been developed on top of these APIs and evaluated on a range of datasets belonging to different tasks. A summary of datasets used in FrugalGPT LLM cascade experiments is given in Table II. Fig. 6 shows the FrugalGPT strategy.

Dataset	Domain	Size	No. of Examples in the prompt
HEADLINES	Finance	10000	8
OVERRULING	Law	2400	5
COQA	Passage Reading	7982	2

TABLE II: Summary of datasets used in the FrugalGPT LLM cascade experiments.

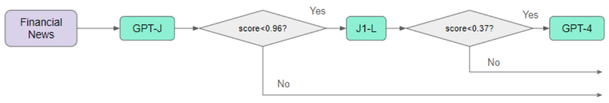


Fig. 6: FrugalGPT Strategy

Approach	Accuracy	Cost($)
GPT-4	0.857	33.1
FrugalGPT	0.872	6.5

TABLE III: Overall performance and cost

Using a cascade strategy learned from the dataset with an overall budget of $6.5, which is one-fifth of GPT-4's cost, FrugalGPT demonstrates a smart approach. It avoids querying GPT-4 if it gets high-quality answers from GPT-J and J1-L. Even when GPT-4 makes an occasional mistake, FrugalGPT learns to rely on the correct answers from J-1 and GPT-J. In summary, FrugalGPT impressively cuts costs by 80%

while enhancing accuracy by 1.5% compared to GPT-4. This showcases the potential of FrugalGPT as a cost-effective and accurate alternative in the context of LLMs [13]. Overall performance and cost of FrugalGPT compared to GPT-4 LLM API is given in Table III.

F. Techniques of Checking similarity

1) Cosine Similarity: Cosine similarity is a way to figure out how alike two vectors are [29]. It focuses on the direction or angle of the vectors, not their size. For this to work, both vectors should be from the same inner product space, meaning they can give a single number when multiplied together [25]. The similarity between the vectors is determined by looking at the cosine of the angle between them. This helps us understand how much they point in the same direction, regardless of how long or short they are [27]. Mathematically, the product of two vectors' lengths divided by their dot product can be defined as the cosine similarity. Formula (1) is used to find the similarity between two vectors called A and B.

$$Similarity(A, B) = cos(\theta) = \frac{A.B}{||A||\ ||B||} \tag{1}$$

Cosine similarity yields values within the range of -1 to +1, with greater values indicating a higher degree of similarity. This calculation involves the dot product of the vectors divided by the product of their magnitudes. In practical terms:

- When two vectors share the same orientation, resulting in a 0-degree angle, the cosine similarity is 1.
- Perpendicular vectors, forming a 90-degree angle, yield a cosine similarity of 0.
- Vectors pointing in opposite directions, creating a 180-degree angle, have a cosine similarity of -1.

2) Jaccard Similarity: Jaccard Similarity serves as a measure to gauge how similar two sets are [30], [31]. Whether it's comparing binary vectors or sets of items, this index, denoted as J, helps determine the degree of resemblance. Ranging from 0 to 1, a value closer to 1 signifies more similarity between the two sets. Commonly used in Data Science and Machine Learning applications like Text Mining and Recommendation Systems, Jaccard Similarity is calculated by considering the number of shared observations in both sets divided by the total number in either set. Mathematically, it's expressed as the intersection of sets A and B divided by their union. If two datasets share all the same elements, their Jaccard Similarity Index is 1, indicating complete similarity; if there are no common elements, the index is 0. Essentially, Jaccard Similarity provides insights into how much features overlap between datasets.

G. Cache Eviction Policies

Cache eviction policies are like decision-making tools for handling data in a cache, which is a speedy and temporary storage layer. This cache helps boost performance by holding onto recently used or frequently accessed data in locations that are quicker and more efficient to get to than regular memory spaces. But when the cache reaches its limit, these policies step in to decide which items should be removed to make space for new ones. They essentially manage the revolving door of data in the cache to keep things running smoothly.

1) Least Recently Used (LRU): The item that hasn't been used in a while is removed according to the LRU cache eviction policy. Assuming that the most frequently accessed items will likely be requested again soon, the idea is to keep them in the cache. LRU can be implemented using a linked list. In a linked list, each node represents a cached item. Most recently used item is the head of the list. An item comes to the top of the list when it is accessed. An item is taken out of the tail of the list each time it needs to be removed [32].

2) Least Frequently Used (LFU): The least-used item is removed from the cache using the LFU eviction policy. Assuming that the most popular items are more likely to be requested again soon, the idea is to keep them in the cache. To implement LFU, each cached item can have a frequency counter, and the items can be stored in ascending order of frequency using a priority queue or a hash map of doubly linked lists. An item's position in the data structure is updated and its frequency counter is increased each time it is accessed. Every time an item needs to be removed, it is removed from the group with the lowest frequency [32].

LFU stands out as a more sophisticated cache eviction policy compared to LRU, offering solutions to some of LRU's challenges. It's adept at preventing cache pollution by prioritizing the removal of items accessed infrequently, regardless of when they were last used. LFU also excels in handling cyclic access patterns, favoring items accessed more often than their counterparts. However, these advantages come with trade-offs. Maintaining LFU can be costlier than LRU, especially if frequent updates to frequency counters and data structures are required. Additionally, LFU may suffer from aging, favoring older items with high frequencies even if they've become irrelevant. It may also struggle in applications with changing access patterns, as items popular in the past may not align with current or future trends. In essence, LFU offers a refined approach to cache management, addressing certain limitations while introducing considerations for effective implementation [33].

3) First In First Out (FIFO): FIFO serves as a straightforward cache eviction policy, prioritizing the removal of the item that entered the cache earliest. This policy operates on the premise that retaining the most recently added items in the cache enhances the likelihood of their imminent reuse. FIFO is implemented by using a queue data structure. The oldest item is at the front of the queue and that node represents a cached item. The back of the line is where new items are queued, because of the fact that they are cached. The item at the front of the queue, which represents the earliest addition to the cache, is dequeued when the cache fills up and an eviction is required. FIFO provides a simple and intuitive approach to cache management, aligning with the assumption that recently added items are more prone to future access [34].

4) Time To Live (TTL): TTL functions as a cache eviction policy designed to remove items that have exceeded their predefined duration for staying in the cache. The key concept is to prioritize the retention of the most current items in the cache, operating under the assumption that they hold greater relevance and utility compared to outdated counterparts. The implementation of TTL involves assigning a timestamp to each cached item, alongside a priority queue or a hashmap of doubly linked lists to organize items based on their expiration times. Upon adding an item to the cache, its timestamp is set to the current time plus the TTL value, indicating the duration it is allowed to remain in the cache. When eviction becomes necessary, the item with the earliest expiration time is removed. TTL offers a dynamic approach to cache management by ensuring the preservation of timely and pertinent information while discarding items that have surpassed their designated lifespan [35].

III. Methodology

Initially, we collected course materials and curated datasets to serve as the base for our system. Subsequently, we planned the data flow within the system and designed a high-level solution architecture to guide our implementation. Fig. 7 shows the data flow diagram and Fig. 8 shows the High level solution architecture. The core of our system lies in the Question Answering (QA) model, where we acquired pre-built models, trained them using our datasets, and defined a threshold value (0.5) to identify the highest-scored answer. For that, we selected the most accurate QA model called BERT-base-cased-squad. To enhance the contextual understanding, we incorporated a Similarity Checker utilizing a cosine similarity function, enabling the selection of the most relevant passage for subsequent QA processing. The Cache Implementation employed a Least Frequently Used (LFU) Eviction Policy, replacing least frequently accessed prompts and optimizing storage efficiency. Backend Implementation involved encoding prompts using the all-MiniLM-L6-v2 model, interfacing with the cache, similarity checker, and QA model. Frontend Implementation focused on creating an intuitive chat application using Angular. The Integration phase harmonized the various components, including frontend, backend, cache, similarity checker, and QA model implementations. Finally, extensive testing was conducted to assess the robustness of the chat application across a spectrum of prompts.

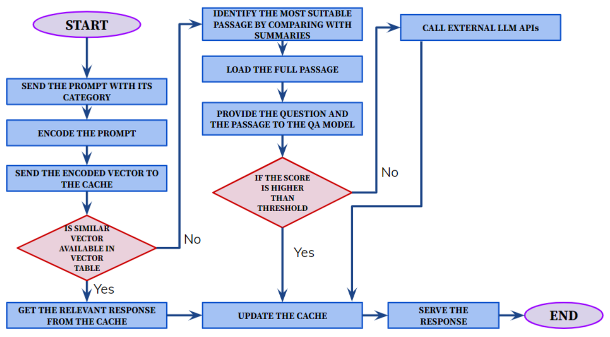

START
SEND THE PROMPT WITH ITS CATEGORY
ENCODE THE PROMPT
SEND THE ENCODED VECTOR TO THE CACHE
IS SIMILAR VECTOR AVAILABLE IN VECTOR TABLE
No
Yes
GET THE RELEVANT RESPONSE FROM THE CACHE
IDENTIFY THE MOST SUITABLE PASSAGE BY COMPARING WITH SUMMARIES
LOAD THE FULL PASSAGE
PROVIDE THE QUESTION AND THE PASSAGE TO THE QA MODEL
IF THE SCORE IS HIGHER THAN THRESHOLD
No
Yes
CALL EXTERNAL LLM APIs
UPDATE THE CACHE
SERVE THE RESPONSE
END

Fig. 7: Data flow diagram.

The user initiates the process by submitting a prompt with its designated category, representing a question directed to our system. We employed an all-MiniLM-L6-v2 encoder sourced from the Hugging Face Sentence Transformer to encode the prompt into a vector. The encoded vector is then forwarded to the cache, where we determine if similar questions have been previously posed to our system. In the event of a cache hit, the relevant response is retrieved from the cache, and the cache's access count is updated. We employ cosine similarity for instances without a cache hit to identify the most suitable passage among summaries generated from the original learning materials. This selected passage is retrieved from the file storage. In its non-encoded form, the original question and the passage are then provided to the question-answering model, specifically BERT-base-uncased-squad2, known for its lightweight yet effective performance. The model identifies the position of the answer in the passage and assigns a score representing the probability of correctness. Setting a threshold value 0.5, determined through trial and error, guides decision-making. If the score surpasses the threshold, the cache is updated with the response from the question-answering model, and the user is served the response. Conversely, external Large Language Model (LLM) APIs are called if the score is below the threshold. In our system, we created a mock service for these APIs rather than invoking external APIs, ensuring efficient and controlled testing procedures.

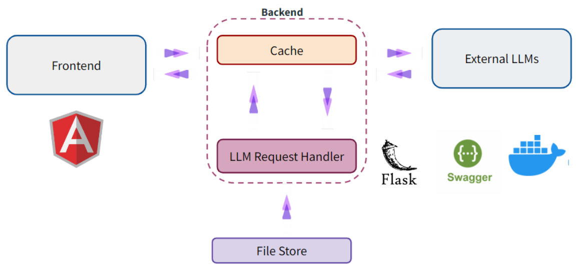


Fig. 8: High Level Solution Architecture diagram.

In crafting the high-level solution architecture for our system, we designed a user interface through an Angular framework, shaping a chat application to facilitate seamless interactions. The backend of our system is built on Flask, a micro web framework, serving as the central hub for managing cache operations and orchestrating Large Language Model (LLM) requests. The backend plays a dual role, directly communicating with LLM APIs for external queries and establishing a unidirectional link with the file storage system. This connection enables the retrieval of pertinent passages required for generating accurate responses. This architectural arrangement ensures a well-structured and efficient system, providing a user-friendly interface while handling the intricate caching processes, LLM interactions, and data retrieval.

IV. Experiments

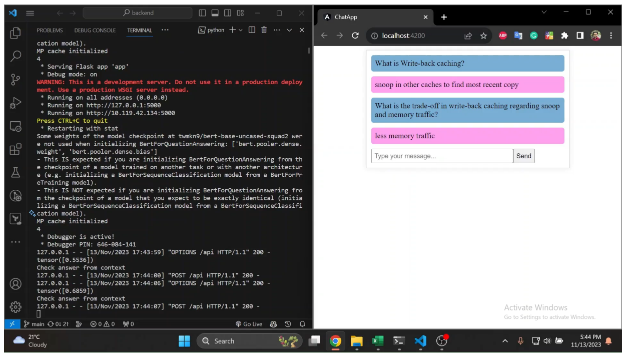
Our research comprised two distinct experiments, each focusing on different implementations. In the "Question Answering Model" implementation, we applied both prebuilt question-answering models and custom models tailored specifically to the context of computer architecture. Custom models, including BERT, Electra-Base, RoBERTa, DistilBERT, DistilRoBERTa, Electra-Small, and XLNet, enhanced the system's performance in Natural Language Processing (NLP) tasks. The custom models, including prebuilt and context-specialized ones, demonstrated notable performance in the question-answering system. The BERT Base Cased model, in particular, exhibited the highest accuracy. The highest achieved was approximately 0.268.

In the Cache Implementation, we explored the effectiveness of the Least Frequently Used (LFU) eviction policy. Our analysis involved testing access counts when retrieving similar questions, ensuring the cache's responsiveness to repeated queries. Additionally, we examined the cache's ability to adapt to new questions and answers, affirming its dynamic nature by replacing the least frequently used block.

The cache implementation's evaluation of the LFU eviction policy highlighted its efficiency in managing storage, optimizing access counts, and ensuring the adaptability of the cache to evolving question-response pairs. These results underscore the robustness of our system in effectively handling a range of question-answering tasks within the unique domain of computer architecture. Further analyses and optimizations can build upon these findings, enhancing the system's overall performance and applicability in diverse educational contexts.

- Steps of the demonstration

1) How to send prompts and get responses



```
cation model).
MP cache initialized
4
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deploy
ment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.119.42.134:5000
Press CTRL+C to quit
 * Restarting with stat
Some weights of the model checkpoint at twmkn9/bert-base-uncased-squad2 wer
e not used when initializing BertForQuestionAnswering: ['bert.pooler.dense.
weight', 'bert.pooler.dense.bias']
- This IS expected if you are initializing BertForQuestionAnswering from th
e checkpoint of a model trained on another task or with another architectur
e (e.g. initializing a BertForSequenceClassification model from a BertForPr
eTraining model).
- This IS NOT expected if you are initializing BertForQuestionAnswering fro
m the checkpoint of a model that you expect to be exactly identical (initia
lizing a BertForSequenceClassification model from a BertForSequenceClassifi
cation model).
MP cache initialized
4
 * Debugger is active!
 * Debugger PIN: 646-084-141
127.0.0.1 - - [13/Nov/2023 17:43:59] "OPTIONS /api HTTP/1.1" 200 -
tensor([0.5536])
Check answer from context
127.0.0.1 - - [13/Nov/2023 17:44:00] "POST /api HTTP/1.1" 200 -
127.0.0.1 - - [13/Nov/2023 17:44:06] "OPTIONS /api HTTP/1.1" 200 -
tensor([0.6859])
Check answer from context
127.0.0.1 - - [13/Nov/2023 17:44:07] "POST /api HTTP/1.1" 200 -
```

What is Write-back caching?

snoop in other caches to find most recent copy

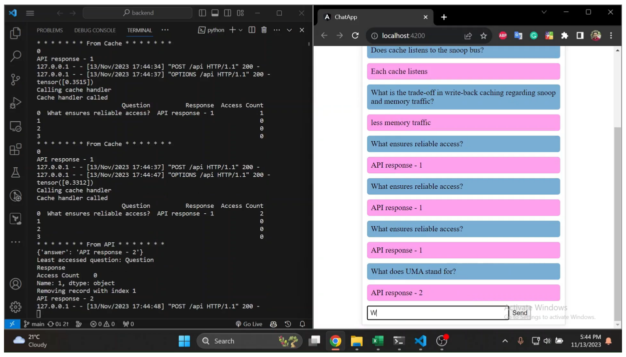
What is the trade-off in write-back caching regarding snoop and memory traffic?

less memory traffic

Fig. 9: Sending prompts and getting responses

The prompts get the responses from the local context. It can be seen in the backend terminal.

2) How do the prompts get responses from external API while it updates the cache. Then after that, prompts will get the response from the cache again.
3) When a user inserts similar types of questions (same question but different sentences), It will give the same response on the cache.
4) How blocks of the cache are getting filled. We have four blocks in our cache. All of the blocks are getting filled.



```
* * * * * * * From Cache * * * * * * *
0
API response - 1
127.0.0.1 - - [13/Nov/2023 17:44:34] "POST /api HTTP/1.1" 200 -
127.0.0.1 - - [13/Nov/2023 17:44:37] "OPTIONS /api HTTP/1.1" 200 -
tensor([0.3515])
Calling cache handler
Cache handler called
                        Question          Response  Access Count
0  What ensures reliable access?  API response - 1             1
1                                                              0
2                                                              0
3                                                              0
* * * * * * * From Cache * * * * * * *
0
API response - 1
127.0.0.1 - - [13/Nov/2023 17:44:37] "POST /api HTTP/1.1" 200 -
127.0.0.1 - - [13/Nov/2023 17:44:47] "OPTIONS /api HTTP/1.1" 200 -
tensor([0.3312])
Calling cache handler
Cache handler called
                        Question          Response  Access Count
0  What ensures reliable access?  API response - 1             2
1                                                              0
2                                                              0
3                                                              0
* * * * * * * From API * * * * * * *
{'answer': 'API response - 2'}
Least accessed question: Question
Response
Access Count    0
Name: 1, dtype: object
Removing record with index 1
API response - 2
127.0.0.1 - - [13/Nov/2023 17:44:48] "POST /api HTTP/1.1" 200 -
```

Does cache listens to the snoop bus?

Each cache listens

What is the trade-off in write-back caching regarding snoop and memory traffic?

less memory traffic

What ensures reliable access?

API response - 1

What ensures reliable access?

API response - 1

What ensures reliable access?

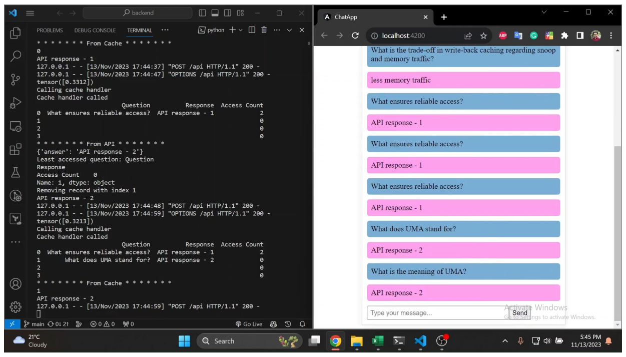
API response - 1

What does UMA stand for?

API response - 2

W

Fig. 10: Calling external LLM APIs



```
* * * * * * * From Cache * * * * * * *
0
API response - 1
127.0.0.1 - - [13/Nov/2023 17:44:37] "POST /api HTTP/1.1" 200 -
127.0.0.1 - - [13/Nov/2023 17:44:47] "OPTIONS /api HTTP/1.1" 200 -
tensor([0.3312])
Calling cache handler
Cache handler called
                        Question          Response  Access Count
0  What ensures reliable access?  API response - 1             2
1                                                              0
2                                                              0
3                                                              0
* * * * * * * From API * * * * * * *
{'answer': 'API response - 2'}
Least accessed question: Question
Response
Access Count    0
Name: 1, dtype: object
Removing record with index 1
API response - 2
127.0.0.1 - - [13/Nov/2023 17:44:48] "POST /api HTTP/1.1" 200 -
127.0.0.1 - - [13/Nov/2023 17:44:59] "OPTIONS /api HTTP/1.1" 200 -
tensor([0.3213])
Calling cache handler
Cache handler called
                        Question          Response  Access Count
0  What ensures reliable access?  API response - 1             2
1       What does UMA stand for?  API response - 2             0
2                                                              0
3                                                              0
* * * * * * * From Cache * * * * * * *
1
API response - 2
127.0.0.1 - - [13/Nov/2023 17:44:59] "POST /api HTTP/1.1" 200 -
```

What is the trade-off in write-back caching regarding snoop and memory traffic?

less memory traffic

What ensures reliable access?

API response - 1

What ensures reliable access?

API response - 1

What ensures reliable access?

API response - 1

What does UMA stand for?

API response - 2

What is the meaning of UMA?

API response - 2

Fig. 11: Cache hits of similar questions

5) How the block replacement happens using the LFU eviction policy. The block is replaced by a new question with a response. It is replaced with the least frequently used block in the cache.

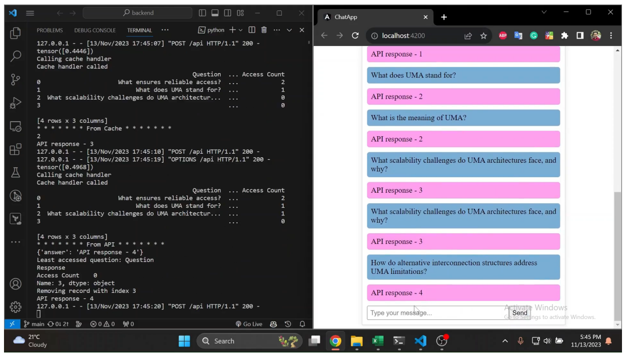
V. Results and Discussion

This section revolves around an in-depth cost analysis, revealing a substantial 70% reduction achieved through our implemented system. This reduction is solely based on cost considerations, excluding accuracy assessments for simplicity. However, the complexity of cost reduction when accuracy is factored in is acknowledged, paving the way for future work. We delve into the cost analysis, highlighting the methodology employed, the specific percentage reduction achieved, and the potential implications of incorporating accuracy metrics into the evaluation framework. This sets the stage for a detailed exploration of our findings, providing a comprehensive understanding of the cost-effectiveness of our system.

A. *Results*

B. *Discussion*

In this work, we propose a cost-effective intelligent tutoring system, demonstrated through implementing a prototype designed to showcase the reduction in costs associated with external Large Language Model (LLM) API usage. Central to our approach is integrating a cache system with the question-answering model, strategically minimizing reliance on external LLM APIs like GPT-3.5. For the demonstration, we selected



```
127.0.0.1 - - [13/Nov/2023 17:45:07] "POST /api HTTP/1.1" 200 -
tensor([0.4446])
Calling cache handler
Cache handler called
                                            Question  ... Access Count
0                     What ensures reliable access?  ...            2
1                          What does UMA stand for?  ...            1
2  What scalability challenges do UMA architectur...  ...            0
3                                                    ...            0

[4 rows x 3 columns]
* * * * * * * From Cache * * * * * * *
2
API response - 3
127.0.0.1 - - [13/Nov/2023 17:45:10] "POST /api HTTP/1.1" 200 -
127.0.0.1 - - [13/Nov/2023 17:45:19] "OPTIONS /api HTTP/1.1" 200 -
tensor([0.4968])
Calling cache handler
Cache handler called
                                            Question  ... Access Count
0                     What ensures reliable access?  ...            2
1                          What does UMA stand for?  ...            1
2  What scalability challenges do UMA architectur...  ...            1
3                                                    ...            0

[4 rows x 3 columns]
* * * * * * * From API * * * * * * *
{'answer': 'API response - 4'}
Least accessed question: Question
Response
Access Count    0
Name: 3, dtype: object
Removing record with index 3
API response - 4
127.0.0.1 - - [13/Nov/2023 17:45:20] "POST /api HTTP/1.1" 200 -
```

API response - 1

What does UMA stand for?

API response - 2

What is the meaning of UMA?

API response - 2

What scalability challenges do UMA architectures face, and why?

API response - 3

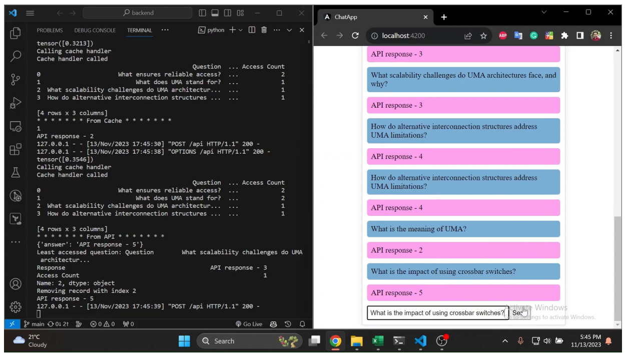
What scalability challenges do UMA architectures face, and why?

API response - 3

How do alternative interconnection structures address UMA limitations?

API response - 4

Fig. 12: Cache blocks filling



```
tensor([0.3213])
Calling cache handler
Cache handler called
                                            Question  ... Access Count
0                     What ensures reliable access?  ...            2
1                          What does UMA stand for?  ...            1
2  What scalability challenges do UMA architectur...  ...            1
3  How do alternative interconnection structures ...  ...            1

[4 rows x 3 columns]
* * * * * * * From Cache * * * * * * *
1
API response - 2
127.0.0.1 - - [13/Nov/2023 17:45:30] "POST /api HTTP/1.1" 200 -
127.0.0.1 - - [13/Nov/2023 17:45:38] "OPTIONS /api HTTP/1.1" 200 -
tensor([0.3546])
Calling cache handler
Cache handler called
                                            Question  ... Access Count
0                     What ensures reliable access?  ...            2
1                          What does UMA stand for?  ...            2
2  What scalability challenges do UMA architectur...  ...            1
3  How do alternative interconnection structures ...  ...            1

[4 rows x 3 columns]
* * * * * * * From API * * * * * * *
{'answer': 'API response - 5'}
Least accessed question: Question        What scalability challenges do UMA
 architectur...
Response                                        API response - 3
Access Count                                                   1
Name: 2, dtype: object
Removing record with index 2
API response - 5
127.0.0.1 - - [13/Nov/2023 17:45:39] "POST /api HTTP/1.1" 200 -
```

API response - 3

What scalability challenges do UMA architectures face, and why?

API response - 3

How do alternative interconnection structures address UMA limitations?

API response - 4

How do alternative interconnection structures address UMA limitations?

API response - 4

What is the meaning of UMA?

API response - 2

What is the impact of using crossbar switches?

API response - 5

What is the impact of using crossbar switches?

Fig. 13: Block replacement in cache

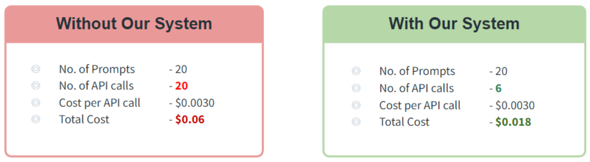
the GPT-3.5 Turbo LLM API, recognized for its capabilities and cost-effectiveness, particularly with a 16K context window optimized for dialog [37], [38].

Model	Input	Output
gpt-3.5-turbo-1106	\$0.0010/ 1K tokens	\$0.0020/ 1K tokens
gpt-3.5-turbo-instruct	\$0.0015/ 1K tokens	\$0.0020/ 1K tokens

TABLE IV: Price Structure of GPT 3.5 LLM API

In our cost analysis, GPT-3.5 Turbo's pricing structure, based on tokens (roughly 750 words per 1000 tokens), was considered. Table IV shows the price structure of GPT 3.5. The API charges \$0.0010 per 1000 tokens for input and \$0.0020 per 1000 tokens for output. Without our system, utilizing 20 prompts resulted in a total cost of \$0.06 for 20 API calls. However, utilizing the local context for answers, our system significantly reduced the number of external LLM API calls to only 6, each incurring a cost of \$0.0030 per API call. Consequently, the total cost for API calls with our system amounted to \$0.018 [37].

This demonstration led to a noteworthy conclusion: our system achieves a 70% reduction in costs, a metric calculated as (profit / total cost from GPT-3.5) x 100%. Notably, this analysis focused solely on cost considerations, paving the way for an exploration when accuracy is factored in a dimension that holds complexity and warrants future work. The algorithmic integration of cost and accuracy metrics promises a



Without Our System	
No. of Prompts	- 20
No. of API calls	- 20
Cost per API call	- \$0.0030
Total Cost	- \$0.06

With Our System	
No. of Prompts	- 20
No. of API calls	- 6
Cost per API call	- \$0.0030
Total Cost	- \$0.018

$$\textit{Cost Reduction} = \frac{\$0.06 \; - \; \$0.018}{\$0.06} \text{ x } 100 = 70\ \%$$

Fig. 14: Cost Analysis

comprehensive evaluation framework for further refinement and optimization of our intelligent tutoring system.

VI. Conclusion

In conclusion, the implemented solution addresses the challenges posed by high costs in utilizing Large Language Models (LLMs) within educational platforms. The devised prototype, equipped with a cache, stands out for its adaptability to integrate seamlessly with various course materials, featuring a custom-based memory instead of a static one. Through systematic experimentation, we implemented the efficacy of our solution in cost reduction by minimizing the reliance on external LLM APIs. By taking advantage of the finding that users typically ask the same kinds of queries, our prototype achieves multiple cache hits, significantly reducing the number of API calls and, consequently, a noteworthy decrease in costs. While our solution focused on computer architecture course materials, the versatility of our prototype allows it to be effortlessly integrated into any course. The success of this approach highlights its potential for widespread adoption, offering a scalable and cost-effective solution for educational platforms leveraging LLMs.

References

[1] M. Wooldridge, "Intelligent agents," Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, vol. 1, pp. 27-73, 1999.

[2] D. L. Poole and A. K. Mackworth, "Artificial Intelligence: Foundations of Computational Agents," Cambridge University Press, 2010.

[3] A. Vaswani et al., "Attention is All You Need," in Advances in Neural Information Processing Systems, vol. 30, 2017.

[4] G. Prato, E. Charlaix, and M. Rezagholizadeh, "Fully Quantized Transformer for Improved Translation," 2019.

[5] Y. Chang et al., "A Survey on Evaluation of Large Language Models," arXiv preprint arXiv:2307.03109, 2023.

[6] T. H. Kung et al., "Performance of ChatGPT on USMLE: Potential for AI-assisted Medical Education Using Large Language Models," PLoS Digital Health, vol. 2, no. 2, p. e0000198, 2023.

[7] A. Caines et al., "On the Application of Large Language Models for Language Teaching and Assessment Technology," arXiv preprint arXiv:2307.08393, 2023.

[8] Y. Liu et al., "Summary of ChatGPT-Related Research and Perspective Towards the Future of Large Language Models," Meta-Radiology, p. 100017, 2023.

[9] X. Q. Dao, "Performance Comparison of Large Language Models on Vnhsge English Dataset: OpenAI ChatGPT, Microsoft Bing Chat, and Google BERT," arXiv preprint arXiv:2307.02288, 2023.

[10] J. G. Meyer et al., "ChatGPT and Large Language Models in Academia: Opportunities and Challenges," BioData Mining, vol. 16, no. 1, p. 20, 2023.

[11] Y. Zhou et al., "Large Language Models are Human-Level Prompt Engineers," arXiv preprint arXiv:2211.01910, 2022.

[12] K. Greshake et al., ”More than You’ve Asked For: A Comprehensive Analysis of Novel Prompt Injection Threats to Application-Integrated Large Language Models,” arXiv preprint arXiv:2302.12173, 2023.

[13] L. Chen, M. Zaharia, and J. Zou, ”FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance,” arXiv preprint arXiv:2305.05176, 2023.

[14] A. Bozkurt et al., ”Speculative Futures on ChatGPT and Generative Artificial Intelligence (AI): A Collective Reflection from the Educational Landscape,” Asian Journal of Distance Education, vol. 18, no. 1, 2023.

[15] V. Fernoagă et al., ”Intelligent Education Assistant Powered by Chatbots,” eLearning & Software for Education, vol. 2, 2018.

[16] A. Przegalińska, ”Collaborative Artificial Intelligence: The Example of Virtual Assistants and Conversational AI,” Artificial Intelligence (AI) as a Megatrend Shaping Education, p. 12, 2022.

[17] P. Sridhar et al., ”Harnessing LLMs in Curricular Design: Using GPT-4 to Support Authoring of Learning Objectives,” arXiv preprint arXiv:2306.17459, 2023.

[18] S. S. Gill et al., ”Transformative Effects of ChatGPT on Modern Education: Emerging Era of AI Chatbots,” Internet of Things and Cyber-Physical Systems, vol. 4, pp. 19-23, 2024.

[19] D. Baidoo-Anu and L. Owusu Ansah, ”Education in the Era of Generative Artificial Intelligence (AI): Understanding the Potential Benefits of ChatGPT in Promoting Teaching and Learning,” SSRN 4337484, 2023.

[20] C. H. Chang and G. Kidman, ”The Rise of Generative Artificial Intelligence (AI) Language Models—Challenges and Opportunities for Geographical and Environmental Education,” International Research in Geographical and Environmental Education, vol. 32, no. 2, pp. 85-89, 2023.

[21] H. Yu and Y. Guo, ”Generative Artificial Intelligence Empowers Educational Reform: Current Status, Issues, and Prospects,” in Frontiers in Education, vol. 8, p. 1183162, June 2023.

[22] C. Cao, ”Scaffolding CS1 Courses with a Large Language Model-Powered Intelligent Tutoring System,” in Proceedings of the International Conference on Intelligent User Interfaces (IUI), Mar. 2023, pp. 229-232, doi: 10.1145/3581754.3584111.

[23] E. Kasneci et al., ”ChatGPT for Good? On Opportunities and Challenges of Large Language Models for Education,” Learning and Individual Differences, vol. 103, p. 102274, 2023.

[24] O. Levy and Y. Goldberg, ”Dependency-Based Word Embeddings.”

[25] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger, ”From Word Embeddings To Document Distances.”

[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, ”BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” Oct. 2018. [Online].

[27] C. Aguerrebere, I. Bhati, M. Hildebrand, M. Tepper, and T. Willke, ”Similarity Search in the Blink of an Eye with Compressed Indices,” Apr. 2023.

[28] D. Narayanan et al., ”Cheaply Evaluating Inference Efficiency Metrics for Autoregressive Transformer APIs,” arXiv preprint arXiv:2305.02440, 2023.

[29] Xia, P., Zhang, L. and Li, F., 2015. Learning similarity with cosine similarity ensemble. Information sciences, 307, pp.39-52.

[30] G. I. Ivchenko and S. A. Honov, ”On the Jaccard Similarity Test,” Journal of Mathematical Sciences, vol. 88, pp. 789-794, 1998.

[31] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, ”Using of Jaccard Coefficient for Keywords Similarity,” in Proceedings of the International MultiConference of Engineers and Computer Scientists, vol. 1, pp. 380-384, March 2013.

[32] D. Lee et al., ”On the Existence of a Spectrum of Policies That Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies,” in Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, May 1999, pp.

[33] P. R. Jelenković and A. Radovanović, ”Least-Recently-Used Caching with Dependent Requests,” Theoretical Computer Science, vol. 326, no. 1-3, pp. 293-327, 2004.

[34] J. Yang, Y. Zhang, Z. Qiu, Y. Yue, and R. Vinayak, ”FIFO Queues Are All You Need for Cache Eviction,” in Proceedings of the 29th Symposium on Operating Systems Principles, October 2023, pp. 130-149.

[35] N. C. Fofack, P. Nain, G. Neglia, and D. Towsley, ”Performance Evaluation of Hierarchical TTL-Based Cache Networks,” Computer Networks, vol. 65, pp. 212-231, 2014.

[36] M. A. K. Raiaan et al., ”A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges,” 2023.

[37] OpenAI Pricing. [Online]. Available: https://openai.com/pricing

[38] L. Spangher et al., ”Officelearn: An OpenAI Gym Environment for Reinforcement Learning on Occupant-Level Building’s Energy Demand Response,” in Tackling Climate Change with Artificial Intelligence Workshop at NeurIPS, 2020.