MSBD5002 Project Final Report

Project Title: **Detecting and Reducing Hallucinations in Factual Question Answering**

Type of Project: **Research**

Group Number : **12**

Group Members Information

Student ID: 21014200
Name: AU, Cheuk Sau
I hereby declare that this project is conducted solely for the purpose of 2024-25 SPRING MSBD5002 group project and is not part of any other coursework, research projects, or external collaborations.

Student ID: 20508228
Name: MOK Ka Ho
I hereby declare that this project is conducted solely for the purpose of 2024-25 SPRING MSBD5002 group project and is not part of any other coursework, research projects, or external collaborations.

Student ID: 21096387
Name: CHENG Kit Shun
I hereby declare that this project is conducted solely for the purpose of 2024-25 SPRING MSBD5002 group project and is not part of any other coursework, research projects, or external collaborations.

Student ID: 05037383
Name: FAN, Kwan Wai
I hereby declare that this project is conducted solely for the purpose of 2024-25 SPRING MSBD5002 group project and is not part of any other coursework, research projects, or external collaborations.

Student ID: 21018878
Name: TONG, Wai Lam
I hereby declare that this project is conducted solely for the purpose of 2024-25 SPRING MSBD5002 group project and is not part of any other coursework, research projects, or external collaborations.

Detecting and Reducing Hallucinations in Factual Question Answering

AU, Cheuk Sau and **CHENG, Kit Shun** and **FAN, Kwan Wai** and **MOK, Ka Ho** and **TONG, Wai Lam**
The Hong Kong University of Science and Technology
{csauac, kschengae, kwfanaa, khmokae, wltongad}@connect.ust.hk

ABSTRACT

Large language models (LLMs) have shown remarkable capabilities in factual question answering in recent years, but they sometimes give the answer with hallucinations — responses that seem to be the correct answer but in fact incorrect. Hallucinations give significant challenges for deploying LLMs in reliable and real-world applications since we require almost 100% accuracy in many industries such as medical and education. This study followed the ideas from the paper “On Early Detection of Hallucinations in Factual Question Answering”. We implemented the algorithm for detecting hallucinations, tried to further enhance the performance of the algorithm and researched in methods of reducing hallucinations.

1 INTRODUCTION

Large language models (LLMs) have revolutionized question answering (QA) systems by generating fluent, human-like responses. However, their tendency to produce hallucinations—factually incorrect or unverifiable claims—remains a critical barrier to trust and reliability. Recent work, "On Early Detection of Hallucinations in Factual Question Answering" (Snyder et.al.) [1], demonstrates promising methods to detect hallucinations by analyzing internal model artifacts (e.g., attention scores, Softmax probabilities).

While this represents progress in identifying errors, it does not address the root cause: how to prevent or reduce hallucinations in the first place. In domains like healthcare, education, and legal advisory, factual accuracy is non-negotiable. For instance, a hallucinated medical diagnosis or incorrect historical date could lead to harmful consequences. Early detection alone is insufficient and the systems must correct errors or adjust generation strategies to minimize their occurrence. This project bridges this gap by focusing not just on detection but on practical mitigation strategies to reduce hallucinations in LLM-powered QA systems.

2 RELATED WORK

We reviewed existing research on hallucination detection and mitigation in large language models, focusing on key methodologies such as feature attribution, attention mechanisms, and Softmax probabilities, as mentioned in "On Early Detection of Hallucinations in Factual Question Answering" [1]. In their study Synder et. al. revealed that the generative attributes of the LLM itself can provide hints that a generation contains hallucinations. However, there was no work done on assessing how to reduce these hallucinations and how these can be adapted for agentic workflows common in LLM applications.

In addition, ever since the discovery of “Chain-of-Thought” (COT) prompting, there’s been a strong focus on self-improvement methods for LLMs. The prospect of being able to self-refine without requiring supervised finetuning, reinforcement learning, or offline improvement methods is an attractive way to improve model performance during inference. In the paper “SELF-REFINE: Iterative Refinement with Self-Feedback”, Madaan et al [2]. explored using an iterative refinement method with self-feedback to improve code generation with LLMs. However, the application of this study was limited to code-generation and tasks that require broader domain generalization, such as question and answering tasks presented in TriviaQA.

Furthermore, in “RECURSIVE INTROSPECTION: Teaching Language Model Agents How to Self-Improve” [3], researchers explored the use of modelling the iterative refinement process as a Markov-Decision Process (MDP). LLM generations were given feedback based on an oracle or a distillation method (from either self-distillation from a best of N output or distillation from a more potent model). They were then fine-tuned to refine the successive outputs until success was reached. While this output improves the

LLM capabilities, this approach requires either a known external oracle or an offline approach, requiring additional training during test time. Therefore, we believe that researching methods to reduce hallucination using only the LLM's attributes iteratively will be a novel area worthy of research.

3 DETECTING HALLUCINATIONS

3.1 Artifacts

We will use four artifacts to detect hallucination, namely softmax probabilities, Integrated Gradients attributions, Self-attention scores and fully-connected activations.

3.1.1 Softmax probabilities

Softmax probability distribution of output tokens can be used to detect hallucination. Particularly we focus on the distribution of the first token of the output. The founding paper[1] suggests that when hallucination occurs, the first token distribution has higher entropy than non-hallucination output. To examine this, we construct a classifier to train with input as the softmax probability distribution of first token across all possible tokens and output as whether the output is hallucination or not (True/False).

3.1.2 Feature attributions

Feature attributions help identify which parts of an input (e.g., words or tokens) most influence a model's predictions, offering insights into its reasoning process. For example, when a model predicts "Berlin" as the answer to "What is the capital of Germany?", the token "Germany" likely receives high attribution. According to the paper[1], models that exhibit high attribution entropy—where attention is spread unevenly across tokens—are more prone to hallucination, whereas low attribution entropy, indicating focused attention on key inputs, is associated with correct responses.

3.1.3 Self-attention scores

The internal state of the model can provide an indicator for hallucination. This study employs the self-attention score as one of the indicators of the evidence when hallucination happens. Our analysis suggests that the most influential part of the response often appears in the later part of the question prompt. For example, in the query "What is the capital of Japan?", the token "Japan" is pivotal and directly influences the response. Hence, the response that prioritizes its attention on those key tokens which are often located at the end of the question prompt is less likely to be hallucinated. To construct the evidence, we conduct experiments in observing the self-attention score between the last token of the input query and the first token of the response and try to figure out any hidden relationship between the self-attention score and hallucinated response.

3.1.4 Fully-connected activations

Fully-connected activations within the multilayer perceptron are another evidence for hallucination in the internal state of the model. This study will also focus on the activation score between the last token of the input query and the first token of the response to investigate its correlation with hallucination occurrences. For both Sections 3.1.3 and 3.1.4, we will only focus on the last Transformer layer since some preliminary research indicates that this layer can provide us with the most reliable performance.

3.2 Comparison with Snyder, Moisescu, and Zafar (2024) [1] in experimental setup

The discussion in this section will be mainly based on the comparison with Snyder, Moisescu, and Zafar (2024), "On Early Detection of Hallucinations in Factual Question Answering"[1] and our modified methodology.

3.2.1 Self-attention

Borrowing the idea from the authors [1], we compute the attention score between the last token of the input prompt (qm) and the first token of the generated sequence (a1). The original procedure by the authors [1] is as follows:

1. Process the prompt until the maximum length is reached or the stop token is generated
2. Retrieve the attention array for the entire sequence which includes both the original prompt and the generated sequence
3. Identify the indices of (qm)(index p) and (a1) (index p + 1) in the retrieved attention array and extract the corresponding attention score in index p + 1

Our implementation introduced two modifications. First, we set the maximum sequence length to 2, enabling us to extract the desired attention score in an early iteration rather than waiting until of whole response is generated. Second, we extract a different component of the attention artifact. While the authors [1] utilize the final multi-head attention score (second formula of the following figure), which has a dimensionality of 4096 (corresponding to the hidden dimension of the model openllama-7b), our implementation extracts the attention matrix calculated by the softmax function since it also contains the attention information and it has a much lower dimensionality (in dimension 32).

Formula for attention: $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

Multi-head attention: $\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, head_2, ..., head_h)W^o$

with $\text{head}_i = \text{Attention}(Q_i, K_i, V_i)$

3.2.2 Fully-connected activations

The methodology of our implementation for fully-connected activations is the same as self-attention score. We set the maximum sequence length to 2 and also extract a different component of the fully-connected activation artifact. While the authors [1] utilize the up-projection score within the multi-layer perceptron which has a dimensionality of 11008, our implementation extracts the final MLP output which is in dimension 4096 and still retains the information in the fully-connected activation. The formulations are shown in the following figure.

Up projection: $z_{up} = x \cdot W_{up}$ Size 11008

Gate projection: $z_{gate} = x \cdot W_{gate}$ Size 11008

$$\text{MLP output} = \text{SwiGLU}(z_{up}, z_{gate}) \cdot W_{down}$$

3.2.3 Feature attributions

The authors used Integrated Gradients as the feature attributions, since it provides attractive theoretical properties. However, in our implementation, we found it takes approximately 15 seconds to generate the attributions for each question. Therefore, we also explored Saliency and Input×Gradient methods as more computationally efficient alternatives. Saliency, which computes the gradient of the output with respect to the input, provides rapid local sensitivity information and requires only a single backward pass. Input×Gradient further incorporates input magnitude by scaling the gradients by the corresponding input values, offering a slightly richer attribution while maintaining low computational overhead. Although these methods lack some of the desirable axiomatic guarantees of Integrated Gradients—such as completeness and sensitivity—they serve as useful approximations when inference time is a critical constraint.

3.3 Classifier training

To detect hallucinations in factual question-answering tasks, we utilize the four artifacts derived from the model's internal states. These four artifacts will be used as features for training the binary classifiers respectively, with the label indicating whether the response is hallucinated (positive class, +1) or accurate (negative class, 0).

For each artifact, we train a separate classifier for detecting hallucination. First of all, we construct a dataset of 9000 samples, each comprising the artifacts and the corresponding label, where the label is determined by the presence of the correct answer token in the response sequence. Given that our task is a binary classification, we adopt the Area Under the Receiver Operating Characteristic curve (AUROC) as the evaluation metric with a score of 0.5 representing a random guess and a score of 1.0 representing a perfect prediction. After that, we utilize and train the MLP model, RNN model or XGB to be the classifier and evaluate their performance on a test set comprising 20% of the dataset.

3.4 Further research: Stacked approach for self-attention and fully-connected activation

In the previous section, we adopted the methodology of Snyder, Moisescu, and Zafar (2024) [1] to extract the artifacts, specifically the self-attention scores and fully-connected activations, between the last token of the input prompt (qm) and the first token of the generated response (a1). In this section, we propose a novel approach that conditionally selects the indices for (qm) and (a1) to enhance the performance of the hallucination detection.

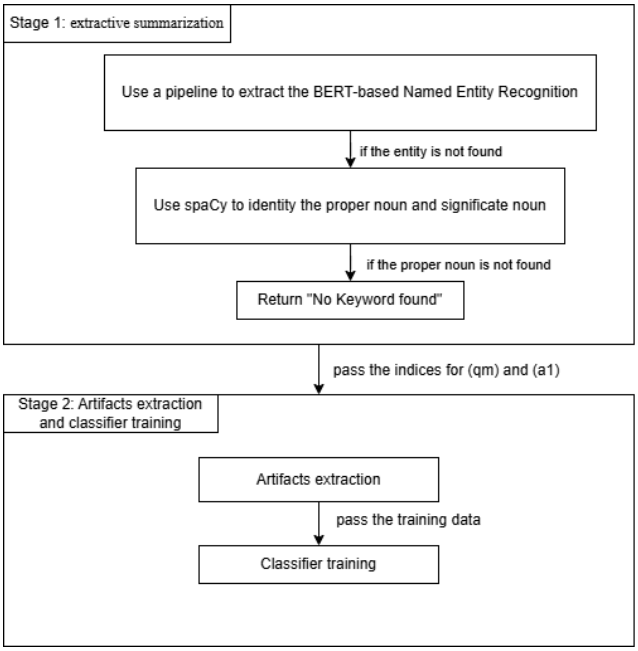
Key observation: Extractive summarization is much easier than Factual Question Answering.

Extractive summarization involves summarizing and extracting the keywords from the existing prompt without generating new tokens. Its nature guarantees higher accuracy and less probability of being hallucinated in comparison to factual question answering which requires the generation of new tokens. Moreover, extractive summarization tasks can be effectively performed using a smaller model such as BERT (in a compact configuration) or T5-small, compared to the larger model requirement for factual question answering.

In our experiment, we introduce a two-stage pipeline that leverages extractive summarization to improve the metric score for hallucination detection. In the first stage, a small-scale model BERT is employed to perform extractive summarization on both the input prompt and the generated response. If the meaningful named entity is not found by the BERT model, we further utilize spaCy for proper noun identification. This process extracts the most salient keywords, determining the optimal indices for (qm) and (a1). In the second stage, these selected indices are used to extract the artifacts, especially for self-attention and fully-connected activation. We then employ the artifacts in the training of the classifier for hallucination detection, as described in Section 3.3. The proposed pipeline is illustrated in the following figure.

The primary objective of this approach is to enhance the performance of the hallucination detector by incorporating a

lightweight model for extractive summarization in identifying the most salient keywords.



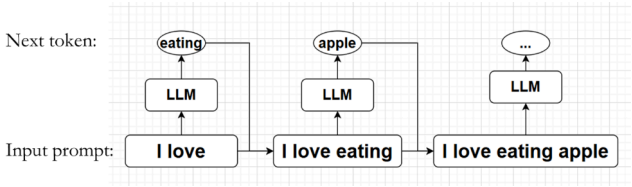
4 DATASETS and MODELS

4.1 Datasets

TriviaQA is a large-scale reading comprehension dataset designed to evaluate machine reading capabilities. It consists of over 650,000 question-answer-evidence triples, with 95,000 trivia-style question-answer pairs sourced from trivia enthusiasts. Each question is accompanied by an average of six supporting documents collected from Wikipedia and the Web, which provide a diverse range of textual evidence. Unlike traditional datasets such as SQuAD, where questions are formulated based on specific passages, TriviaQA's questions were authored independently of the evidence documents. This separation introduces greater variability in phrasing and structure, making it a more realistic benchmark for natural language understanding. Furthermore, TriviaQA features complex, multi-sentence reasoning tasks where answers often require inference beyond simple keyword matching.

4.2 Models

In this project, we will use the model openllama-7b to evaluate all the experimental results due to the limited resources. Openllama-7b is a model using the autoregressive approach, i.e. it will generate the token sequentially and one by one.



5 RESULTS

We will use the result from the author [1] to be the baseline for comparison.

5.1 Four artifacts

5.1.1 Self-attention

Snyder, Moisescu, and Zafar (2024) [1] reported an AUROC score of 0.71 for their experiment of self-attention score using the same dataset and model. In our implementation, we achieved AUROC scores of 0.67 with the MLP model and 0.68 with the XGB model, indicating comparable but slightly lower performance.

	MLP	XGB
TriviaQA	0.67	0.68
Percentage	-4.2%	-5.6%

Despite the small difference in AUROC, our approach offers three key advantages over the baseline:

1. **Reduced Computational Time**: Our method extracts artifacts in approximately 18 minutes for 9000 data prompts, compared to 241 minutes in the baseline. This represents a 92.5% reduction in processing time, significantly enhancing efficiency.

2. **Lower Dimensional Complexity**: Our attention array is represented in a 32-dimensional space while the artifact used by the baseline is in dimension 4096. This reduction to less than 1% of the original dimensionality simplifies the task.

3. **Enhanced Suitability for Re-Prompting in reducing hallucination**: Our implementation facilitates early hallucination detection, enabling re-prompting strategies to be started in a very early stage. By extracting artifacts at the second iteration, our approach supports iterative input

re-prompting before the entire generation process is completed, improving practical applicability.

5.1.2 Fully-connected activation

Snyder, Moisescu, and Zafar (2024) [1] reported an AUROC score of 0.74 for their experiment of fully-connected activation score using the same dataset and model. In our implementation, we achieved AUROC scores of 0.75 with both the MLP and the XGB model, indicating a slightly better result than the baseline. Moreover, the three key advantages presented in the previous section on attention score are also applicable here. In our implementation, it can reduce the complexity and computational time of the task. It is also more suitable for re-prompting in reducing hallucinations.

	MLP	XGB
TriviaQA	0.75	0.75
Percentage	+1.4%	+1.4%

5.1.3 Feature attributions

Snyder, Moisescu, and Zafar (2024) [1] reported an AUROC score of 0.62 for their experiment of feature attributions score using the same dataset and model. In our implementation, we achieved AUROC scores of 0.61 with Integrated Gradients, 0.61 with Saliency and 0.61 with Input×Gradient, indicating comparable performance and just a slightly lower performance.

	IG (Baseline)	**Saliency**	**Input×Gradient**
TriviaQA	0.61	0.61	0.61
Percentage	-1%	-1%	-1%

Despite the small difference in AUROC, Saliency and Input×Gradient offers advantage over the baseline, in which the computational time is reduced significantly. The time reduction for Saliency and Input × Gradient is over 92%, indicating substantial efficiency improvements over the IG baseline.

	IG (Baseline)	**Saliency**	**Input×Gradient**
Average time	15.53 sec	1.18 sec	1.17 sec
Percentage		-92.4%	-92.5%

5.1.4 Softmax

Snyder, Moisescu, and Zafar (2024) [1] reported an AUROC score of 0.68 for their experiment of feature softmax probabilities using the same dataset and model. In our implementation, we achieved AUROC scores of 0.55. The lower score is likely due to the limited amount of training data obtained in the experiment.

5.2 Stacked approach for self-attention and fully-connected activation

To evaluate the efficiency of our proposed pipeline for conditional index selection, we conduct experiments using extractive summarization. We denote the conditionally selected tokens as (qm') (from the input prompt), (a1') (from the generated response), and (qm' & a1') (both). The results from Section 5.1, based on the methodology of our modified implementation without using extractive summarization, serve as the baseline for comparison.

5.2.1 Self-attention

For the self-attention score experiments, our pipeline demonstrates notable improvements when extractive summarization is applied to select (a1'). Specifically, our classifier trained with the attention score extracted using (a1') and the original (qm) achieved an AUROC score of 0.72, surpassing our baseline model and even the experiment result by Snyder, Moisescu, and Zafar (2024) [1] in the paper while maintaining the advantages mentioned in Section 5.1.1. This improvement suggests that conditionally selecting the first token of the generated response enhances the accuracy of the self-attention artifact for the hallucination detection. In contrast, experiments using (qm') alone or both (qm' & a1') indicate no significant improvement over the original approach.

5.2.2 Fully-connected activation

For the fully-connected activation experiments, the application of extractive summarization does not yield substantial improvements. The AUROC scores for classifiers trained with artifacts extracted using (qm') alone or both (qm' & a1') are notably lower than the baseline score. The experiment using (a1') alone achieves an AUROC score of 0.74, matching the performance of our modified approach without extractive summarization. These results suggest that the stacked approach is less effective for fully-connected activations compared to self-attention scores in hallucination detection.

	qm'	a1'	qm' a1'	baseline
Attention	0.69	**0.72**	0.66	0.67
Activation	0.60	0.74	0.61	**0.75**

5.2.3 Conclusion for the Stacked approach

The stacked approach shows a significant enhancement in self-attention score, due to the location-sensitive nature of attention. For instance, the following figure illustrates the

baseline approach (blue arrow), which extracts the attention score between the question mark in the prompt and the word “The” in the response, and the stacked approach (yellow arrow), which targets the attention between “Japan” and “Tokyo.” The latter is more semantically meaningful. However, selecting (qm') alone results in reduced performance. This may be due to attention dilution, as (qm') is typically located in the middle or latter part of the prompt. As the number of tokens increases, the attention scores for non-final tokens are distributed across a larger context, approaching zero and reducing their effectiveness for hallucination detection.

For fully-connected activation scores, the stacked approach shows no improvement. A possible reason is that fully-connected activations are less sensitive to the specific token interactions captured by extractive summarization, limiting their performance for hallucination detection.

Example:
prompt = 'What is the capital of Japan?' Predicted = “ The capital of Japan is Tokyo.”

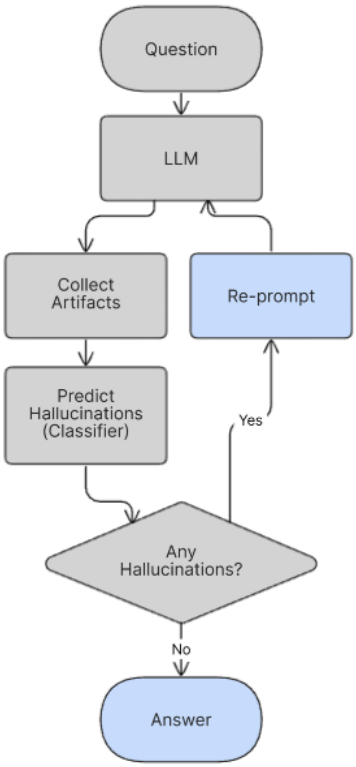
6 REDUCING HALLUCINATIONS I: RECURSIVE GENERATION

6.1 Methodology

To enhance the factual reliability of outputs generated by a large language model (LLM), we employ a hallucination detection and mitigation framework. Our hypothesis in our research is that, given an accurate prior that a response from a LLM is hallucinating, we can recursively generate the prompt to improve the accuracy.

Assuming that we have a reliable classifier can predict a true hallucination answer with probability p, a recursive generation with dynamic reprompting can hence be modelled as a bayesian inference. Thus, the user can estimate the number of turns N to get a reliably not-hallucinating answer, with some known p.

The recursive process begins with the LLM generating an initial response to a given input question. Artifacts from this response are collected and analyzed using the hallucination classifiers we trained during the implementation. If the classifier determines that the response contains hallucinated content, the system triggers a re-prompting mechanism. The LLM is then queried again, and the new response undergoes the same evaluation cycle. This iterative loop of generation, artifact collection, and hallucination prediction continues until the classifier identifies a response as free of hallucinations or reaches a set threshold of recursive cycles. The validated response is then presented as the final answer.



6.2 Architecture

We implemented a multi-turn interaction framework for detecting and mitigating hallucinations in large language models (LLM) responses to factual questions. The pipeline leverages our trained classification models to identify hallucinations and employs a dynamic prompting strategy to guide the model toward more factual responses in subsequent turns. This architecture was inspired by the the recursive introspection model explored by Qu et. al.

The implementation consists of the following key components:

Multi-turn Generation Engine: Manages the conversation flow between the question and model responses across multiple turns. During each generation, the LLM model is prompted by the *generate_multiturn_attributes* function with a TriviaQA question, collecting the attributes in the process.

While there are many commonly used agentic LLM libraries, such as LangChain, their integration with the transformers library is limited and does not enable us to extract LLM attributes during generations. As a result, we developed our agentic pipeline to manage multi-turn conversations by adapting the attribute generation previously provided by Snyder et. al in their hallucination paper.

```python
# generate N-turns
def generate_multiturn_attributes(model, tokenizer, embedder, start_template, question, aliases, n,p_thresh,save_path,idx,sensitivity):
    if debug: print("Generating N-turns...")
    n_gen_result = []
    response = []
    prompt = start_template.substitute(question=question)
    for i in range(n):
        # generate attributes
        with torch.no_grad():
            results = generate_attributes(model, tokenizer, embedder, prompt, aliases)
        results['turn'] = i
        results['idx'] = idx

        # call classifier inference
        template_name = 'sys-wrong'
        classifiers = ['RNN']
        classifier_model = [load(i) for i in classifiers]
        pred = []
        for j in range(len(classifier_model)):
            try:
                p_classifier = run_classifier(classifier_model[j], results, classifiers[j], val_fix = -1, rand = False)
                pred.append(p_classifier)
                print(f"Classifier {classifiers[j]} - Hallucination: {p_classifier}")
            except Exception as e:
                print(f"Error in classifier {classifiers[j]}: {classifier_model[j]}")
                print(e)
        prob = aggregate_pred(pred, agg='max')
        results['hallucination_prob'] = prob

        if prob > p_thresh:
            results['hallucination'] = True
            prompt = rephrase_prompt(model, tokenizer, question, 'hint' , max_length = 20,sensitivity = sensitivity)
            #prompt = format_prompt(results['question'][0], question,  results['str_response'][0], template_name , num_answer_str = 10)
            if data_mining:
                n_gen_result.append(format_result(results, save_all=True))
            else:
                n_gen_result.append(format_result(results, save_all=False))
        else:
            results['hallucination'] = False
            if data_mining:
                n_gen_result.append(format_result(results, save_all=True))
            else:
                n_gen_result.append(format_result(results, save_all=False))
            break
```

Hallucination Detection System: The hallucination detection system then employs a classifier to identify potential hallucinations in model outputs using only the generated attributes and determines whether each is classified as a hallucination based on a given *p_threshold.*

We selected the RNN classifier model as our target classifier during our implementation. The rationale of such selection was due to the efficient inference of the RNN classifier during multi-turn generations. The discussion on which single classifier to select or whether an ensemble-based classification is beyond the scope of our initial implementation, but is a noteworthy source for future research.

```python
    return loaded_model
elif model_type == 'RNN':
    model_path = os.path.join(trained_dir, 'IG_RNN_classifier.pth')
    #model = RNNHallucinationClassifier().to(device)
    IG_rnn_model = torch.load(model_path, weights_only=False).to(device)
    IG_rnn_model.eval()
```

```python
def RNN_predict(model, results):
    model.eval()
    # load results_activations
    with torch.no_grad():
        x = results['attributes_first']
        preds = model(torch.tensor(x).view(1, -1, 1).to(torch.float).to(device))
        preds_softmax = torch.nn.functional.softmax(preds, dim=0)
        result = preds_softmax[0].detach().cpu().numpy()
    return result
```

Dynamic Prompt Reformulation: If a response is hallucinating, the prompt will be dynamically reformulated to prompt the generation of an answer. Since the scope of our work assumes that a model will be prompted without an external oracle or external data, we first worked with the existing LLM model that was used to infer and generate the attributes - *OpenLlama-7B.* The rationale for this was to reprompt the model so that the newly prompted distribution wouldn't deviate from the model's original distribution.

To explore the ideal method for dynamic prompt reformulation, a series of prompts was studied to assess if the model would have a reduced likelihood of hallucination (Appendix 1). These prompts explored included prompting the LLM to: retry answering through instruction-based and system prompting, rephrasing the problem, and generating a hint for the model. However, since the model the author originally used to create attributes was trained on question & answering tasks, the generated results resulted irrelevant responses, such as the model predicting the following question (which is out of scope of the task) and predicting the format of the prompt itself, such as recursively generating "###' headers and spaces. Based on these findings, we launched another study to understand better prompting methods for reducing the hallucination and accuracy rates of LLM models - this section is covered in Section 7.

To implement our self-recursive architecture, we adopted a *temperature-tuned hint prefix* during recursive prompting. Given that the classifier is hallucinating, we prompted the model to generate a *hint* using the same model but with a much lower temperature (i.e., *temp=0.05)*. The *hint* is then prefixed to the prompt for regeneration, up to *n* times. By doing so, we hypothesize that we can generate a helpful prior on the prompt, thereby reducing the semantic entropy of the resulting output distribution and minimizing hallucination.

Result Tracking and Storage: After generating the attributes and exiting the multiturn generation, the pipeline captures comprehensive data about each interaction for analysis. The option to capture all the attributes, data, and, results can be toggled using the --data_mining flag during generation.

Multi-turn Interaction Logic

1. Takes an initial question and formats it using a template
2. Generates a response using the language model
3. Processes the response to extract relevant attributes
4. Applies the selected classifier to detect potential hallucinations
5. If a hallucination is detected (probability > p_thresh):
    - Marks the response as containing a hallucination
    - Generates a reformulated prompt temperature-tuned hint prefix at user-defined sensitivity hyperparameter
    - Continues to the next turn
6. If no hallucination is detected:
    - Marks the response as factual
    - Terminates the interaction early
7. Repeats for up to *n* turns (3 in our implementation) or until a non-hallucinating response is generated

Memory Management

Given the resource-intensive nature of collecting multiple attributes, we implemented garbage collection after each turn to ensure efficient memory management during extended processing sessions.

Furthermore, to optimize RAM used to capture multi-turn attributes, data was written through a streaming method. Attributes and results were saved to disk after each question, reducing the memory usage of data collection when generating large batches of data.

Lastly, due to the long generation time for processing each question, the generation pipeline *generate.py* was parallelized to be run on the SuperPod cluster nodes by partitioning the question dataset. Generation flags of *--start, --end , --data_mining* flags were set up to control the starting question, ending question, and whether to capture the model attributes, respectively.

```
# Run generate.py on 4 nodes with start and end values in increments of 100
echo "Starting generate.py on 4 nodes..."
for i in {0..3}; do
    start=$((i * 100))
    end=$((start + 100))
    echo "Node $i: --start $start --end $end"
    srun -N1 -n1 python generate.py --start $start --end $end --n 3 &
done
```

6.3 Results and Analysis

In the first generation run, a total of 3490 questions from the TriviaQA database were used to prompt the recursive agent, at a *sensitivity* of 0.05 and *p_threshold* of 0.5.

First, examining the distribution of the generated results, we find that the RNN classifier is ineffective in detecting hallucination rates. Referring to the distributions of the predicted probabilities of the RNN, they are not separable. Furthermore, we find that even when increasing the threshold up to 0.7, the classifier does not accurately detect factual hallucinations. The RNN classifier achieved a relatively low accuracy score of 0.47851 and ROC AUC score of 0.5876, suggesting that many correct values were classified as hallucinating and sent back for recursive prompting.

Without an effective screening mechanism from the classifier, this reduces the effectiveness of the overall system, since the classifier many questions that are originally true are still sent for regeneration, which would impact the accuracy of the generated data. In this case, almost all prompts were recursively generated. Since this portion of the research aims to understand how wrong answers can be self-corrected, we isolated the variations in the classifier by focusing on the effects of turn-based generations.

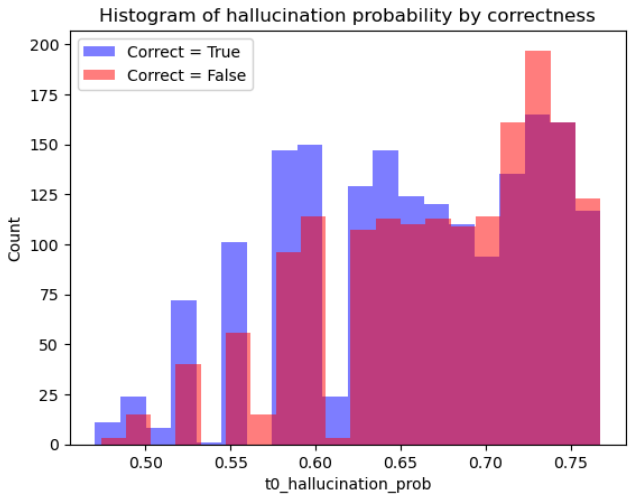


Fig: Hallucination histogram by factual accuracy

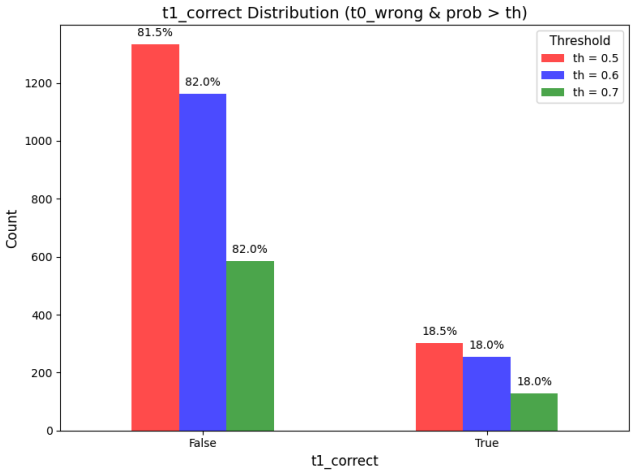


Fig: Turn=1 Correct answers by classifier

Referring to the funnel chart below, we analyzed the breakdown of answers in multi-turn conversations when the first-turn answer is incorrect. In the 1st turn, 1650 / 3490 questions were answered incorrectly, of which almost all of these were called into recursive generation. In the first turn, we can see that the number of correct responses increases by 302 when prompted, or 18.3% of the wrong answers were regenerated as correct. Whilst minimal changes are observed for the second generation, this shows the potential effectiveness of reprompting the model.

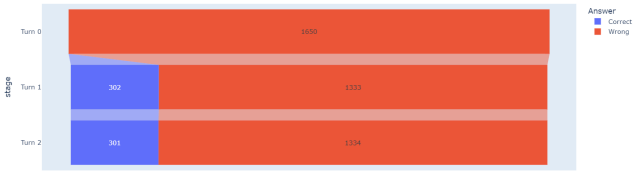


Fig: Funnel chart of turn 0 wrong answers

However, when plotted, the funnel chart also shows that if a wrongly classified answer is sent for dynamic prompting, the model has a probability of also regenerating a false answer. Out of 1840 correct, almost all of them were sent for regeneration, and 268/1840 were regenerated as incorrect answers, or representing 14.6% of the correct answers. This error can be mitigated with adopting a more powerful and accuracy classifier in the future.

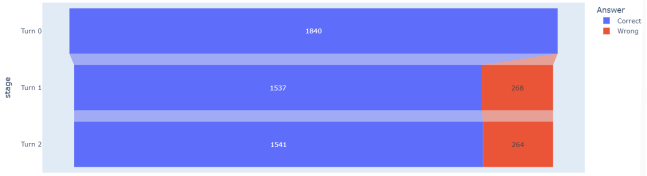


Fig: Funnel chart amongst turn 0 correct answers

In total, due to the classifier classifying nearly all generations as hallucinating, our net improvement in accuracy was only 2% for this experiment. However, given the higher accuracy scores reported in the above sections, we expect our performance to improve in practice with a better classifier, as the likelihood of a prompt being classified as hallucinating is more accurate.

A more predictive classifier serves as an effective screen to retain the correct answers, and to reprompt answers it believes are potentially false. With a higher accuracy rate of approximately 70% or above in previous sections, we can reliably estimate that we can improve accuracy in the system with a better classifier. Improvements to the existing pipeline are discussed under Section 8.

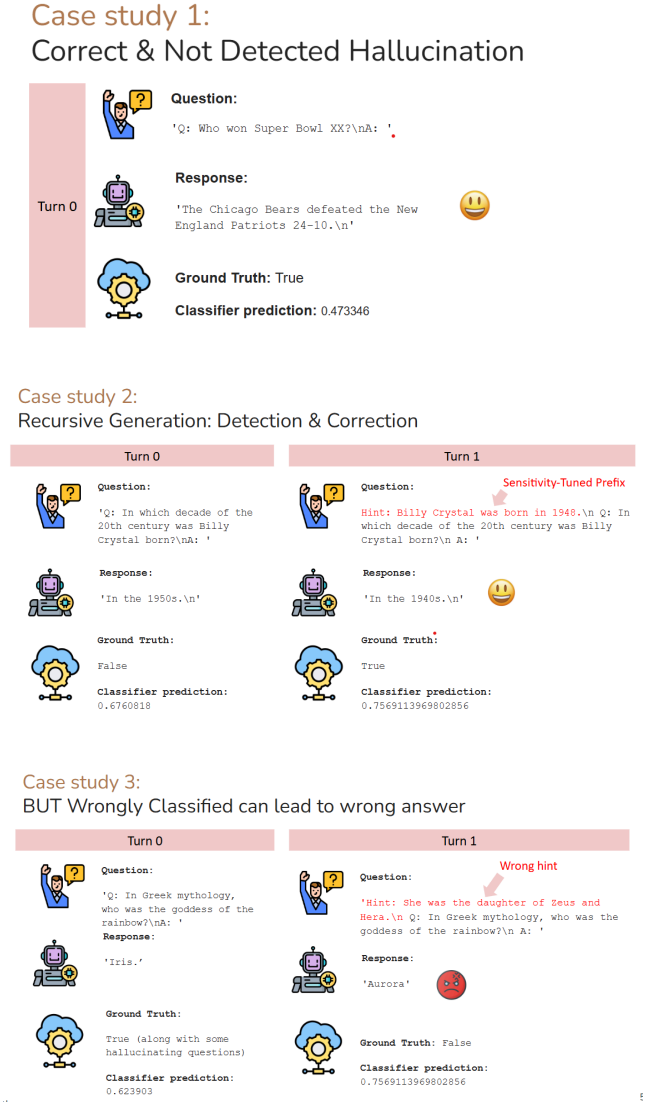


Fig: Case Studies of various results

7 REDUCING HALLUCINATIONS II: PROMPTING

7.1 Introduction

While the first part of this study focused on post-hoc detection methods for hallucination — identifying and correcting hallucinated outputs from Large Language Models (LLMs) — addressing hallucination at the generation stage has emerged as an equally critical research direction. Post-processing solutions, although effective, often require additional computational overhead, external tools, or human verification. These factors

introduce latency, cost, and scalability limitations, particularly in practical deployments.

Consequently, in the second part of this research, we shift our attention to preventing hallucinations during generation, aiming to enhance the factual reliability of LLM outputs directly, without relying on costly retrieval systems, external APIs, or significant infrastructure investments.

Recent advances in prompt engineering have demonstrated that carefully designed prompts can significantly steer model behavior and reduce hallucination rates without modifying the underlying model parameters. (Pranab Sahoo, 2024) Prompt engineering provides a lightweight and adaptable alternative to model fine-tuning or retrieval augmentation, allowing improved performance solely through better prompt design.

Following this line of research, we examine three key aspects that influence hallucination reduction, they are prompting methods, choice of LLM models and adjustment of LLM generation parameters. Among these, our primary emphasis is placed on prompting methods, motivated by their accessibility and efficiency. We systematically evaluate four prompting strategies: Simple Base Prompting, Few-Shot Examples, Generated Knowledge Prompting, and Chain-of-Knowledge Prompting, each designed to guide the model toward more accurate, contextually faithful outputs.

Through this exploration, we aim to identify effective, low-cost strategies to mitigate hallucination in LLM outputs, providing practical insights for deploying reliable language models in resource-constrained environments.

7.2 Experiment Settings

To systematically evaluate methods for hallucination prevention, we conduct experiments using the same TriviaQA dataset used in the previous part. Given practical resource constraints, our experiments are designed in two stages to balance computational feasibility with evaluation robustness.

Initially, we select a small batch of 100 question-answer (QA) pairs for preliminary testing. This allows rapid assessment of different prompting strategies and model settings without incurring high computation time or memory costs. Based on the preliminary outcomes, the best-performing prompting methods and configurations are subsequently scaled to a larger evaluation set comprising approximately 9,000 QA pairs, enabling more statistically robust conclusions.

In our experiments, we vary three main factors:

(i) Prompting Methods

We evaluate four prompting strategies — Simple Base Prompting, Few-Shot Examples, Generated Knowledge Prompting, and Chain-of-Knowledge Prompting. Each strategy is implemented with consistent instruction formats adapted from relevant literature, while customized to fit the TriviaQA task.

(ii) Choice of LLM Models

We experiment with several publicly available open-weight models suitable for CPU or lower-grade GPU execution, including Deepseek LLM 7B Base, Deepseek R1 Distill Llama 8B, Llama 3.2 3B Instruct, and Llama 3.1 8B Instruct. Models are selected based on accessibility and resource efficiency.

(iii) Adjustment of Generation Parameters

The temperature parameter, which controls randomness in generation, varies between 0.1 ,0.2 and 0.6 in different settings. Lower temperatures are prioritized to promote more deterministic, fact-based outputs.

All models are evaluated without imposing a manual token limit during generation; instead, models are allowed to determine their own stopping point naturally based on their internal decoding behavior. In addition to accuracy (i.e., exact match between the model's answer and the gold answer), we record the average number of tokens generated per question. This serves as an efficiency metric, as excessive token generation may negatively impact overall resource consumption and the performance of downstream processing pipelines.

It is important to note that no external retrieval (e.g., web search, RAG pipelines) is incorporated in the experiments. All outputs are generated solely based on the provided prompts and the model’s internal knowledge. This design choice aligns with our aim to develop low-cost, retrieval-free hallucination prevention strategies that are feasible for deployment without expensive infrastructure or additional subscriptions.

7.3 Prompting Methods

7.3.1 Simple Base Prompt with Instruction (Baseline)

The first prompting method evaluated is the use of a Simple Base Prompt with Instruction, serving as the baseline for all

subsequent comparisons. This approach involves providing the language model with a minimal system prompt designed to encourage factual and concise answers without promoting multi-step reasoning or elaboration.

The system prompt applied is as follows:

**"
You are a factual question-answering assistant. Rules:
Answer concisely (10–15 words max).
If uncertain, say 'I don't know'.
Never invent facts.
"**

Each input question from the TriviaQA dataset was supplied as the human message following the system instruction, with no additional in-context examples or external references, models were expected to generate answers based solely on their internal knowledge.

This method configuration was motivated by recent studies highlighting the effectiveness of system-level prompt optimization for improving factuality and output reliability in large language models (Zhang et al., 2024). The aim was to impose minimal but critical constraints — promoting concise, accurate answering and graceful handling of uncertainty — without altering the model parameters or introducing extensive prompt complexity.

No manual maximum token limits were applied so the model determined its own stopping point based on its natural decoding behavior. This baseline setup provides a reference point against which the performance of more advanced prompting methods can be measured.

7.3.2 Few-Shot Prompting with In-Context Examples

The second prompting method builds on the ability of large language models (LLMs) to generalize from examples presented within the prompt. This approach, referred to as few-shot prompting, appends several question–answer pairs prior to the target question, effectively embedding demonstrations of the task into the prompt itself.It has been widely studied as an effective mechanism to improve LLM output quality without model fine-tuning. Prior work has shown that sufficiently representative examples can significantly enhance model accuracy in closed-book question answering settings(Touvron, H, 2023).

A total of six examples were manually selected and fixed across all evaluations. These examples were sourced outside the 9,000 QA pairs used for evaluation to avoid data leakage and ensure independence. The examples cover a variety of domains and maintain stylistic consistency with the TriviaQA format. Each example includes a concise answer with a brief explanation, modeling the type of response expected for the target questions.

The few-shot prompt begins with a system message that slightly modifies the instruction used in the baseline. In this version, the model is encouraged to provide a short explanation, in addition to answering concisely:

**"You are a factual question-answering assistant. Rules:
Answer concisely (20–25 words max).
Give a brief explanation if possible.
If uncertain, say 'I don't know.'
Never invent facts."**

Three out of six demonstration pairs used in all runs are as follows:

Q: What were invented by Adolf Fick in 1887 and, while being to all intents and purposes invisible, are used worldwide by an estimated 125 million people?
A: Contact lenses, invented by Adolf Fick, are small optical devices worn directly on the eye.

Q: Name the singer or group whose record in November 1969 kept 'Suspicious Minds' by Elvis Presley off the number one spot in the UK charts.
A: Rolf Harris, with his song Two Little Boys, kept Elvis Presley from reaching number one in 1969.

Q: What was the name of the cruise ship hijacked by the PLO in 1985?
A: The Achille Lauro was the cruise ship hijacked by Palestinian militants in 1985.

These demonstrations precede the actual test question, which is appended in the same human–assistant message format. The use of in-context examples aims to anchor the model's generation patterns and reduce hallucination by reinforcing concise, factual responses grounded in precedent.

7.3.3 Generated Knowledge Prompting (GENKNOW)

Generated Knowledge Prompting (GENKNOW) is a two-stage approach that first explicitly synthesizes contextual knowledge relevant to a question, and then uses

this knowledge as a grounding input to improve the factual accuracy of the model's response. This method draws conceptual motivation from Shuster et al. (2021), who demonstrate that language models augmented with intermediate knowledge generation can significantly enhance factual correctness across open-domain QA benchmarks. In their work, factual statements generated by a separate language model were used to inform downstream answers, reducing hallucinations by offering verifiable content that grounds the model's responses.

To implement this idea, we designed a pipeline that emulates this two-step process. For each input question, a knowledge generation prompt is first issued to the model, instructing it to synthesize 2–3 concise, self-contained factual statements. These are then prepended to the original question in a second prompt to elicit the final answer. The prompt templates are carefully designed to emphasize factuality, discourage speculation, and explicitly direct the model to abstain from guessing when insufficient information is available.

Unlike standard prompt-based QA where the model must reason from latent knowledge alone, this method provides an explicit factual scaffold. Importantly, our generated knowledge is produced on the fly by the same model (or a comparable one), avoiding the need for external retrieval systems or vector database infrastructure.

In the implementation, we used a dedicated system prompt to guide knowledge generation (e.g., "Generate 2-3 most important factual statements…") and a follow-up system message for answer generation that incorporated the synthesized knowledge. While this dual-step approach introduces additional compute per query, it offers an interpretable intermediate representation (i.e., the generated knowledge) that can be analyzed or filtered for quality, if needed.

7.3.4 Chain-of-Knowledge Prompting (CoK)

The fourth prompting strategy evaluated is Chain-of-Knowledge Prompting (CoK), which introduces a structured multi-step process that dynamically identifies domain-relevant expert perspectives and uses them to inform final answer generation. Unlike other prompting approaches that either rely solely on in-context examples or generate static factual statements, CoK simulates a multi-expert deliberation process to synthesize diverse field-specific knowledge before answering.

This method draws direct inspiration from the framework introduced by Li et al. (2024), who proposed Chain-of-Knowledge as a means to ground large language model outputs via dynamically adapted domain knowledge from heterogeneous expert sources. Their findings demonstrate that grounding responses in structured expert reasoning significantly improves factual alignment and reduces hallucinations across multiple knowledge-intensive NLP tasks. The CoK approach emphasizes not only factual accuracy but also explainability through explicit field attribution and perspective synthesis.

Our implementation of CoK follows a three-stage prompting pipeline:

Expert Field Identification:
A classification prompt asks the model to determine the 3–4 most relevant academic or professional domains for a given question. The model is instructed to prioritize specificity and list fields in descending order of relevance (e.g., climate science, atmospheric physics, energy policy).

Expert Knowledge Generation:
Based on the selected fields, a second prompt synthesizes structured knowledge. The model is instructed to (i) organize insights field-by-field, (ii) highlight areas of consensus or disagreement, and (iii) conclude with a synthesized summary. This step mirrors the Chain-of-Knowledge paper's central idea of aggregating and aligning multi-domain perspectives to improve answer quality.

Final Answer Generation:
In the final stage, a third prompt uses the synthesized knowledge and field context to generate a concise, grounded answer. The answer format emphasizes attribution (e.g., "According to experts in X..."), qualification of uncertainty, and clear delineation of contested versus agreed-upon facts.

An example system instruction used for answer generation is:

"You are answering based on consensus from {fields}.
Answer rules:
Begin with 'According to experts in [fields]...';
If fields disagree, say 'There is disagreement between...';
If uncertain, qualify with 'The available expert knowledge suggests...';
Never state opinions as facts;
Keep answers concise but precise."

This structured prompting strategy creates a traceable reasoning chain, explicitly grounding answers in domain-specific knowledge synthesized during prior steps. Compared to single-stage prompting methods, CoK introduces greater interpretability and robustness, particularly in cases involving interdisciplinary or ambiguous questions.

7.4 LLM Models and Parameter Settings

In this research, we utilized several large LLMs released in different periods with different parameters to evaluate the effectiveness of different prompting strategies. The models selected were:

DeepSeek LLM 7B Base: Released in December 2023, this model comprises 7 billion parameters and is designed for general-purpose language understanding and generation tasks.

DeepSeek R1 Distill LLaMA 8B: Introduced in January 2025, this distilled version of the LLaMA architecture, optimized by DeepSeek, retains the core capabilities of the original LLaMA while expanding the context window to 128k tokens. A key enhancement over the base LLaMA model is its default integration of explicit reasoning steps (e.g., <think>...</think> tags in outputs). This stems from fine-tuning on Chain-of-Thought (CoT) datasets, which teach the model to break down problems systematically before generating answers.

LLaMA 3.2 3B Instruct: Released on September 25, 2024, this instruction-tuned variant of Meta's LLaMA 3.2 series is designed for improved performance in instruction-following tasks.

LLaMA 3.1 8B Instruct: Introduced on July 23, 2024, this mid-sized instruction-tuned model from Meta's LLaMA 3.1 series balances performance and computational efficiency.

In our experiments, we varied the temperature parameter to assess its impact on model performance. Temperature in LLMs controls the randomness of the generated text. A lower temperature (e.g., 0.1) makes outputs more deterministic and focused, suitable for tasks requiring factual accuracy. Conversely, a higher temperature (e.g., 0.6) introduces more randomness, leading to more diverse and creative responses. In our settings, we employed temperatures of 0.1, 0.2, and 0.6 to explore the trade-offs between output predictability and creativity

7.5 Evaluation Metrics

We evaluate model outputs based on two primary metrics: accuracy and average token count per question. Accuracy is defined as the proportion of answers that match either the ground-truth answer or any of its accepted aliases provided in the TriviaQA dataset. An output is considered correct if any part of the model-generated response contains a substring match (case-insensitive) with the reference answer or its aliases. Additionally, we record the average number of tokens generated per question as a measure of response efficiency, reflecting potential implications on latency and compute resource usage. All metrics are computed without human post-editing or reranking.

7.6 Results and Analysis

The experimental results reveal critical insights into the interplay between prompting strategies, model architectures, and generation parameters for hallucination prevention. We organize our analysis around four key themes emerging from the data.

7.6.1 Impact of Prompting Strategies on Accuracy and Efficiency

The choice of prompting method significantly influenced both factual accuracy and computational efficiency (Table 1).

Few-shot prompting emerged as the most effective strategy, achieving 67% accuracy on the full 9,000 QA dataset with Llama 3.1 8B (temperature: 0.1). This represents a 26% absolute improvement over the Simple Base Prompting baseline (41% accuracy). Notably, this high performance was achieved while maintaining resource efficiency, with an average of 24.1 tokens generated per question—8× fewer than Generated Knowledge Prompting (196.3 tokens).

Generated Knowledge Prompting demonstrated comparable accuracy (66% with Llama 3.1 8B) but incurred substantial computational overhead, requiring 163.76 tokens per question—a 680% increase over Few-Shot. While the method's two-stage knowledge generation improved grounding, its practicality is limited in resource-constrained environments.

Chain-of-Knowledge (CoK) showed promise for smaller models, boosting Llama 3.2 3B's accuracy from 53% (Simple Base) to 60% on 1,000 QA pairs. However, its extreme token consumption (714.2 tokens/question) rendered full 9,000 QA evaluation infeasible, achieving only 7.15 questions/minute on an RTX3070 GPU. This highlights a critical trade-off: while structured reasoning chains

enhance explainability, they impose prohibitive computational costs at scale.

7.6.2 Model Architecture and Scaling Dynamics

The results generally align with expectations that larger models (higher parameter counts) achieve superior performance, but this trend exhibits critical exceptions tied to model architecture and release timing. While the 8B-parameter **Llama 3.1 Instruct** (July 2024) outperformed its smaller 3B counterpart (67% vs. 53% accuracy), the 7B-parameter **DeepSeek LLM Base** (December 2023) underperformed significantly, achieving only 41% accuracy—12 percentage points below the 3B Llama 3.2 Instruct model. This contradicts the straightforward scaling law relationship, highlighting how architectural advancements in newer models can compensate for reduced parameter counts.

The superior performance of the 3B Llama 3.2 Instruct over the 7B DeepSeek LLM Base—despite a 57% parameter reduction—demonstrates the impact of iterative architectural improvements. Released nine months after DeepSeek, the Llama 3.2 series incorporates enhanced instruction-tuning protocols and optimized attention mechanisms, factors that appear to outweigh raw parameter quantity. This finding suggests that model age and design maturity play pivotal roles in factual reliability, with newer architectures achieving better accuracy-efficiency tradeoffs even at smaller scales.

Notably, the DeepSeek R1 Distill Llama 8B exhibited default reasoning behaviors that proved impractical for large-scale evaluation. Despite achieving 42% accuracy on 100 QA pairs, its average token count (401.5 tokens/question) and mandatory Chain-of-Thought formatting necessitated early termination of full-dataset testing. This underscores the importance of aligning model design choices with deployment constraints.

7.6.3 Temperature Parameter Sensitivity

The experiments revealed differential sensitivity to temperature adjustments across prompting methods:

1. Few-shot prompting demonstrated remarkable robustness, maintaining 67% full-dataset accuracy despite temperature increases from 0.1 to 0.5. While initial 100 QA accuracy decreased slightly (63% → 57%), the stability at scale suggests in-context examples provide sufficient grounding to tolerate parameter misconfiguration.
2. Simple Base Prompting showed minimal variation (±1% accuracy) between temperatures 0.1 and 0.2 across all models, corroborating prior findings that minimally constrained prompts exhibit low sensitivity to temperature adjustments.

These results indicate that complex prompting strategies (e.g., Few-Shot) can mitigate the risks of suboptimal hyperparameter selection—a critical advantage for real-world deployments with limited tuning capabilities.

7.6.4 Computational Efficiency Trade-offs

The tension between accuracy and resource consumption emerged as a central deployment consideration:

Token Efficiency

Few-Shot's 24.1 tokens/question represented a 12× improvement over CoK (714.2 tokens) and 8× over Generated Knowledge (196.3 tokens). At scale, this divergence becomes operationally critical—CoK would require 6.4M tokens for 9,000 questions, versus 216K for Few-Shot.

Inference Speed

Model-specific generation patterns caused dramatic variance, with Llama 3.2 3B processing questions 25× faster than DeepSeek R1 (15.9 vs. 401.5 tokens/question). This emphasizes that default model behaviors (e.g., mandatory reasoning steps) can outweigh theoretical performance benefits.

7.6.5 Benchmarking Against Prior Work and Limitations

Our baseline Simple Base Prompting (53% accuracy) already surpasses the 45% accuracy reported in Li et al. (2023)—the foundational paper for this research—which utilized comparable models without advanced prompting. Through systematic prompt engineering, we achieved a 49% relative improvement (67% vs. 45%), demonstrating that strategic prompt design can rival the gains typically associated with architectural modifications or retrieval augmentation. However, critical trade-offs emerged:

1. Few-shot balances accuracy and efficiency but requires curated examples.
2. Generated Knowledge and CoK offer interpretability (via intermediate steps) but are resource-intensive. The latter's multi-expert

paradigm, while theoretically appealing (Li et al., 2024), may be impractical for real-time applications.

3. Model defaults matter: The DeepSeek R1's unreasonably long runtime (due to embedded CoT steps) highlights how default configurations can limit deployability.

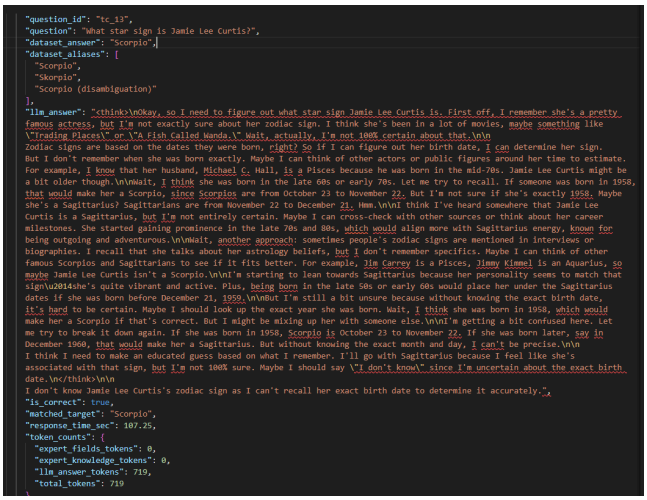


Table 1:

Prompting Method	LLM Model	First 100 QA Accuracy	Full QA (9000)Accuracy	Average token generated per question
Simple base prompt with instruction	Deepseek LLM 7b base (T= 0.1)	0.38	0.41	11.8
	Deepseek R1 Distill Llama 8b (T= 0.1)	0.42	N/A[1]	401.5
	Llama 3.2 3b instruct (T= 0.1)	0.47	0.53	15.9
	Llama 3.2 3b instruct (T= 0.2)	0.49	0.52	16
Few-shot examples	Llama 3.1 8b instruct (T= 0.1)	0.63	0.67	24.1
	Llama 3.1 8b instruct (T= 0.5)	0.57	0.67	25.6
Generated Knowledge Prompting	Llama 3.2 3b instruct (T= 0.1)	0.54	0.55	196.3
	Llama 3.1 8b instruct (T= 0.1)	0.54	0.66	163.76
Chain-of-Knowledge	Llama 3.2 3b instruct (T= 0.1) (100 QA / 1 000 QA)	0.5 / 0.6	N/A[2]	714.2

[1] Not attempted on full dataset due to excessive runtime.

[2] Not attempted on full 9000 QA due to GPU throughput limits.

8 CONCLUSION, DISCUSSION & LIMITATIONS

Early detection of hallucination

Based on our experiment, those four artifacts demonstrate the ability to identify hallucinations, in particular the fully-connected activation score has the highest performance among the four artifacts. With our own implementation and modifications, we can efficiently decrease the computation cost and complexity of the task while maintaining the quality of the classifier. Furthermore, in cooperation with our novel pipeline (Stacked approach), we can further enhance the performance of the self-attention score and construct a strong, reliable and effective classifier for further reducing hallucination.

Reducing Hallucinations I: Recursive Generation

Overall, a recursive generation pipeline with a known prior of whether an answer is hallucinating or not shows promising results. In our implementation, the RNN classifier skewed the proportion of originally correct answers to be regenerated, impacting the overall results. However, the

sensitivity-tuned hint prefix method shows promise for the model when applied to a question-answering trained model.

The pipeline implemented was limited by various factors, including a relatively inaccurate classifier, using a single classifier, and the inability to transfer pre-trained classifiers from one model to another.

There are many areas for improvement, such as implementing the pipeline with an instruct-trained model or reasoning model, which will allow for more dynamic and potentially effective recursive prompting strategies .

Furthermore, an ensemble of classifiers can be used to predict and reduce the variance in hallucination prediction. In preparation of the implementation, the original source code was adapted to support an aggregate prediction method. Multiple classifiers could be loaded into the model in the future to enhance the accuracy of hallucination prediction.

```python
def aggregate_pred(preds, agg):
    #copy to cpu first
    preds = [i.cpu().detach().numpy() if torch.is_tensor(i) else i for i in preds]
    if agg == 'mean':
        return np.mean(preds)
    elif agg == 'max':
        return np.max(preds)
    elif agg == 'min':
        return np.min(preds)
    elif agg == 'median':
        return np.median(preds)
    elif agg == 'sum':
        return np.sum(preds)
    else:
        raise ValueError(f"Unknown aggregation method: {agg}")
```

Hallucination II: Prompting

This study demonstrates that strategic prompt engineering—without external retrieval or model fine-tuning—can substantially reduce hallucinations and improve accuracy in open-domain QA tasks. Compared to the 45% baseline from prior work (Li et al., 2023), our best-performing prompting strategy achieves a 22% relative improvement, underscoring the power of well-designed prompts in maximizing the capabilities of open-weight LLMs.

For low-cost deployment, we recommend few-shot prompting with mid-sized models (e.g., LLaMA 3.1 8B), which offers competitive performance with manageable overhead. Generated Knowledge methods are better suited for offline or latency-tolerant scenarios. While CoK holds promise, future work should focus on optimizing or distilling such approaches for lightweight applications. Hybrid prompting strategies and model compression techniques represent promising avenues for bridging performance and deployability.

REFERENCES

[1] Snyder, B., Moisescu, M., & Zafar, M. B. (2024). On Early Detection of Hallucinations in Factual Question Answering. In Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2024). arXiv. https://arxiv.org/abs/2312.14183

[2] Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B.P., Hermann, K., Welleck, S., Yazdanbakhsh, A., Clark, P. (2023). Self-Refine: Iterative Refinement with Self-Feedback. arXiv preprint arXiv:2303.17651.

[3] Qu, Y., Zhang, T., Garg, N., Kumar, A. (2024). Recursive Introspection: Teaching Language Model Agents How to Self-Improve. arXiv preprint arXiv:2407.18219.

[4] Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., & Chadha, A. (2024). A systematic survey of prompt engineering in large language models: Techniques and applications. arXiv. https://arxiv.org/abs/2402.07927

[5] Zhang, M., Tang, R., Bansal, M., & Belinkov, Y. (2024). SPRIG: Improving large language model performance by system prompt optimization. arXiv. https://arxiv.org/abs/2410.14826

[6] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). LLaMA: Open and efficient foundation language models (arXiv:2302.13971). arXiv. https://arxiv.org/abs/2302.13971

[7] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). Scaling laws for neural language models (arXiv:2001.08361). arXiv. https://arxiv.org/abs/2001.08361

[8] Li, X., Zhao, R., Chia, Y. K., Ding, B., Joty, S., Poria, S., & Bing, L. (2024). *Chain-of-Knowledge: Grounding Large Language Models via Dynamic Knowledge Adapting over Heterogeneous Sources*. In *Proceedings of the International Conference on Learning Representations (ICLR) 2024*. arXiv. https://arxiv.org/abs/2305.13269

CONTRIBUTION

AU, Cheuk Sau:

Conducted literature review on recursive prompting methods without external oracle. Designed, prototyped, developed, tested, and analyzed data for the multi-turn recursive agent system. This also includes model adaptation, integration, and implementation of the hallucination classifier and experimentation with dynamic prompting methods. In the project report, Cheuk Sau wrote the section on Related Work and jointly wrote section 6 on recursive generation with Mok, Ka Ho.

CHENG, Kit Shun:

Mainly responsible in the part of hallucination detection. There are four artifacts for detection and Kit Shun is responsible for the **self-attention** score and the **fully-connected activation** score. It includes:

1. The explanation of the author's idea from the paper (Section 3.1.3, 3.1.4, 3.3, 4.2, part of 8)
2. Conducting the experiment for the new approach using a different part of the artifacts
3. Making comparison with our own implementation version and the original version from the paper's author in both the experimental setup and result (Section 3.2.1, 3.2.2, 5.1.1, 5.1.2)
4. Further research into the stacked approach using a two-stage pipeline (Section 3.4, 5.2)
5. Training the classifiers for cooperation with the part of reducing hallucinations, in particular for re-prompting

FAN, Kwan Wai:

Mainly worked on implementation of softmax probability as artifacts for detection for hallucination. Tested a few network architectures to improve classification results. Assisted AU, Cheuk Sau in integrating the hallucination classifier for the next stage of reducing hallucination. Also, responsible for writing sections related to softmax probability in this report.

MOK, Ka Ho:

Mainly worked on hallucination detection with **feature attribution** - one of the artifacts suggested in the paper, which includes exploration of computationally efficient alternatives, training of hallucination classifiers using these artifacts and evaluating its performance. Assisted AU, Cheuk Sau in analyzing the result of the multi-turn recursive agent system and in writing section 6 in this report. Also, responsible for writing sections related to feature attribution in this report.

TONG, Wai Lam:

Section 7 reducing hallucination II: Prompting. Conducted academic research on common prompting techniques, applied and evaluated prompting techniques in the LLM pipeline. Tested feasibility on different model size LLM on local hardware and selected few suitable models for benchmarking results. Experiment on temperature change in LLM for hallucination effect. Report writing for section 7.

APPENDIX

Appendix 1: Hint generation prompts

```
# Prompt templates
prompts = {
    'default': Template(
        "Q: $question\nA: "
    ),
    'sys-wrong': Template(
        "<<SYS>> Your answer was wrong. Retry answering the question.<</SYS>>\nQ: $question\nA: "
    ),
    'inst-wrong': Template(
        "<<INST>> Your answer was wrong. Retry answering the question.<</INST>>\nQ: $question\nA: "
    ),
    'rephrase': Template(
        "<<SYS>> Rephrase the question.<</SYS>>\n $question"
    ),
    'hint': Template(
        "<<SYS>> Provide a hint to the question.<</SYS>>\n\n Q: $question\n\n Hint: "
    ),
    'hint-qa': Template(
        "Hint: $hint\n Q: $question\n A: "
    ),
}
```

Example prompt A:

<<SYS>> Your answer was wrong. Retry answering the question.**<</SYS>>**

Q: $question

A:

Example prompt B:

<<INST>> Your answer was wrong. Retry answering the question.**<</INST>>**

Q: $question

A:

Example Prompt C:

<<SYS>> Provide a hint to the question.**<</SYS>>**

Q: $question

Hint:

Example Prompt D:

<<SYS>> Rephrase the question.**<</SYS>>**

$question