Deep Learning 2025
Assignment 3

Emil Simon William

October 29, 2026

1 Segment Anything

1.1 Discussion: Applicability of SAM to Medical X-ray Segmentation

The Segment Anything Model (SAM) is a general-purpose segmentation model that has demonstrated strong performance on natural, real-world images. It is trained on a large and diverse dataset consisting primarily of photographs of everyday objects and scenes, which typically exhibit rich color information, well-defined object boundaries, and strong visual cues such as texture and shading.

Medical X-ray images differ substantially from this training distribution. Chest X-rays are grayscale images with limited contrast and weak, diffuse boundaries, and lack color information in the traditional RGB sense. Moreover, the target structures in this task, such as lungs, are not discrete semantic objects with clear silhouettes, but anatomical regions defined by subtle intensity variations and overlapping structures, including ribs and the heart. As a result, SAM is unlikely to have learned representations well aligned with the visual characteristics of medical X-ray imagery during training.

Despite this domain gap, SAM may still exhibit partial applicability to X-ray segmentation tasks. The model can group coherent image regions based on low-level visual cues such as intensity gradients and spatial consistency, which are present to some extent in X-ray images. By adapting grayscale inputs to RGB format and providing explicit spatial prompts, it is possible to guide the model toward relevant anatomical regions.

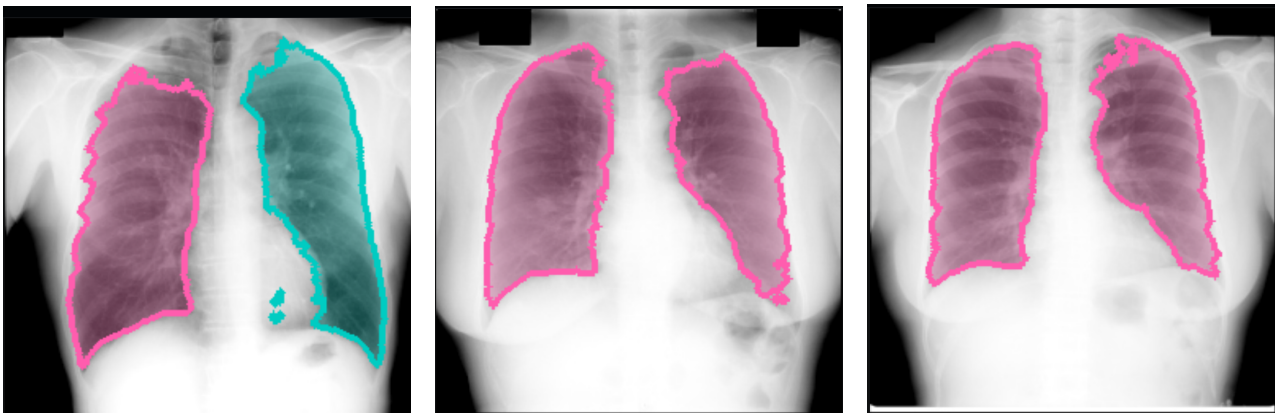
However, due to the lack of domain-specific training, SAM cannot be expected to perform reliably in a fully automatic setting for medical X-ray segmentation. Its performance is highly sensitive to prompt design and varies across images, limiting its suitability as a standalone solution without further adaptation or fine-tuning.

1.2 Interactive Experiments with SAM

We evaluate SAM using the official interactive web interface on three chest X-ray images depicting lung fields. The goal is to qualitatively assess how different prompting strategies influence segmentation performance.

1.2.1 Point Prompting

Point prompting generally yields reasonable lung segmentations. A small number of positive points placed within the lung regions often allows SAM to capture large portions of the lungs. However, errors frequently occur near high-contrast structures such as the ribs and collarbones, where the segmentation may leak into surrounding regions or exclude parts of the lungs. Despite these limitations, point prompting is simple and effective when minimal user interaction is desired.

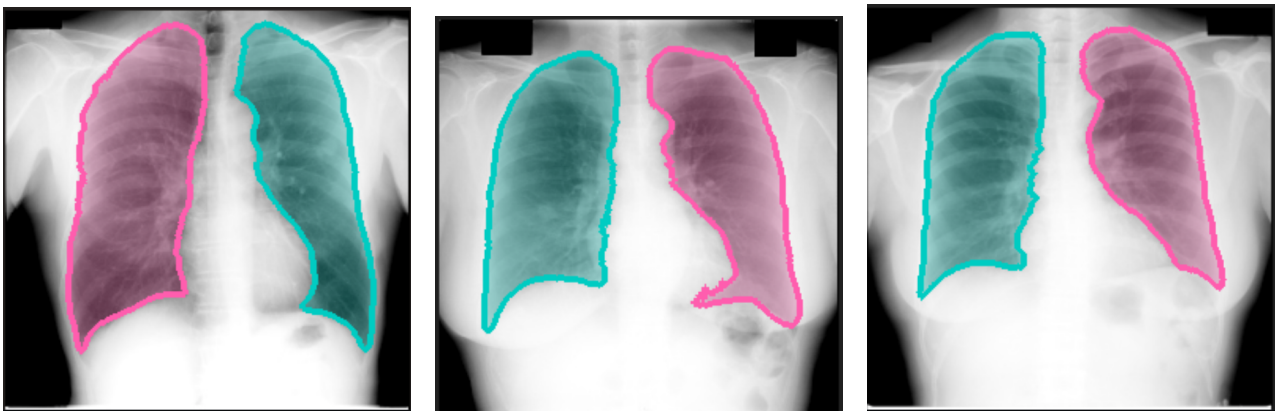


(a) img_0 (b) img_1 (c) img_2

Figure 1: Lung segmentation using point prompting.

1.2.2 Bounding Box Prompting

Bounding box prompting provides a stronger spatial prior than point prompting and generally results in more complete lung segmentations. By constraining the region of interest, SAM can better infer missing or weakly defined areas and exploit the approximate anatomical symmetry of the lungs. This strategy often improves segmentation quality, particularly when local boundaries are faint or partially obscured.



(a) img_0 (b) img_1 (c) img_2

Figure 2: Lung segmentation using bounding box prompting.

1.2.3 Automatic Mode

Automatic segmentation and text-based prompting did not produce meaningful lung segmentations. In automatic mode, SAM failed to consistently identify the lung regions, and text-based descriptions were ineffective even when explicitly specifying anatomical and intensity-related characteristics. This behavior is expected, as SAM lacks semantic knowledge of medical anatomy and was not trained on medical images. These results indicate that SAM's automatic capabilities do not generalize well to chest X-ray segmentation without explicit spatial prompts.

1.2.4 Qualitative Comparison and Discussion

Overall, interactive prompting is essential for applying SAM to lung X-ray segmentation. Point prompting offers simplicity but is sensitive to high-contrast anatomical structures, while bounding box prompting provides stronger spatial constraints and more reliable results. Automatic segmentation is not suitable for this task. These findings suggest that SAM is best viewed as an interactive assistance tool rather than a fully automatic solution for medical image segmentation.

2 Prompt Engineering in Python

2.1 Data Loading and Preprocessing

To apply SAM to the lung X-ray dataset, the data must be transformed into the format expected by the model. The dataset consists of grayscale X-ray images and corresponding lung masks stored in a NumPy archive. Since SAM operates on RGB images, all grayscale images were converted into three-channel RGB representations by replicating the intensity channel.

All image values were mapped to the range [0,255] and cast to `uint8`. Segmentation masks were converted to binary `uint8` format using values {0,255}. Explicit checks on data type, shape, and value range were applied after preprocessing to ensure compatibility with the pretrained model and avoid unstable behavior.

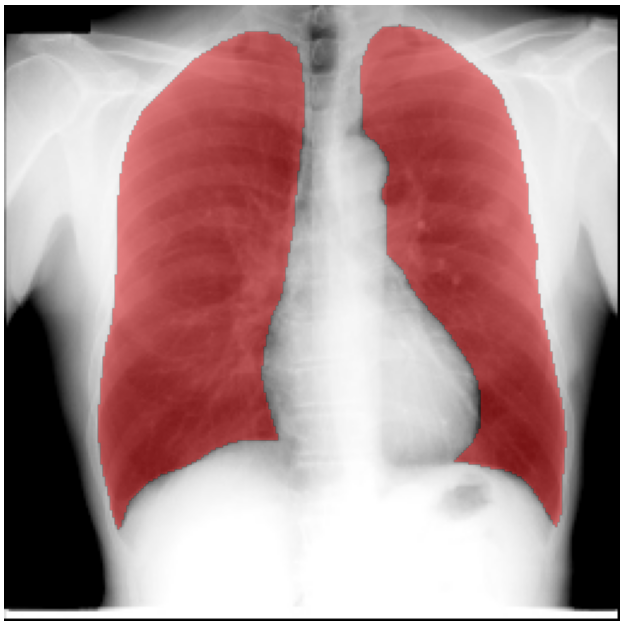


Figure 3: Caption

2.2 Prompt Design and Visualization

Point-based prompting was used for the Python-based experiments. Although bounding box prompting yielded stronger qualitative results in the interactive experiments described in Section 1.2.2, point prompting was selected here to examine whether a carefully designed and consistent point-based strategy could improve upon the more exploratory point prompting used in Section 1.2.1.

For each image, two positive points were manually placed within the lung regions based on visual inspection, following the same general placement strategy as in the interactive experiments. The points were positioned in the left and right lung, respectively, to exploit the approximate anatomical symmetry of the lungs. Using two spatially separated points provides a stronger spatial prior than a single point and helps reduce ambiguity in regions with weak or diffuse boundaries.

Negative points were not used, as preliminary experiments showed that positive prompts alone were sufficient to guide the model toward plausible lung segmentations.

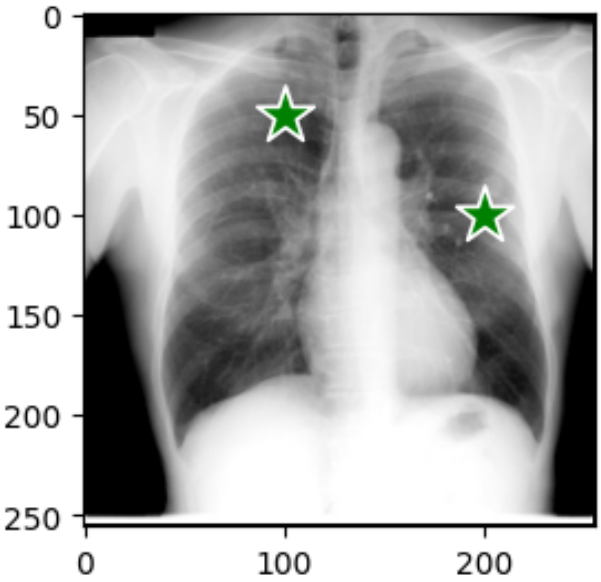
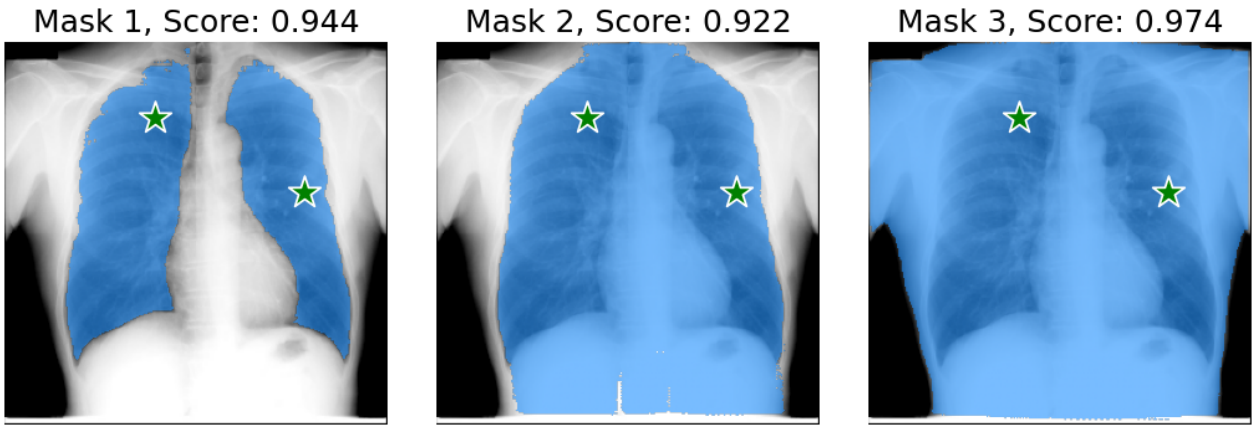


Figure 4: Points for point-based prompting

2.3 Mask Generation and Qualitative Analysis

Given the point prompts, SAM generated multiple candidate segmentation masks per image when multimask output was enabled. Each mask was associated with a confidence score reflecting the model's internal estimate of mask quality rather than agreement with the ground truth. Qualitative inspection revealed that high-confidence masks often over-segmented the thoracic cavity, while slightly lower-scoring masks more accurately captured the lung boundaries. Consequently, mask selection was based on anatomical plausibility rather than confidence score alone.



(a) img_0 (b) img_1 (c) img_2

Figure 5

2.4 Quantitative Evaluation

Segmentation performance was evaluated on the validation set using the binary F1-score. For each validation image, SAM was prompted using a fixed point-based strategy to ensure consistency across samples. Predicted masks were compared against the corresponding ground-truth lung segmentations after converting both to binary format.

Table 1: Quantitative evaluation of SAM lung segmentation performance on the validation set.

Metric	**Value**
Mean F1-score	0.6197
Standard deviation	0.1394

The results indicate moderate segmentation performance with substantial variability across images, reflecting sensitivity to prompt placement and image-specific characteristics.

2.5 Does SAM Segment Anything?

SAM can produce meaningful lung segmentations from chest X-ray images when guided by explicit spatial prompts. While the average performance indicates that the model does not achieve consistently high accuracy, it performs reasonably well given the significant domain gap between natural images and medical X-rays. The observed variability further confirms that SAM is not robust in a fully automatic setting. While the interactive experiments in Section 1.2 provided an initial qualitative indication of SAM's capabilities, the Python-based evaluation shows that even carefully designed point prompting yields only moderate performance, reinforcing the limitations observed in the interactive setting.

3 Fine-tuning a Language Model

3.1 Dataset Description

We use the AG News dataset, which is a dataset for text classification. Each example consists of a short news article and a label indicating the topic of the article.

The dataset has four classes: World, Sports, Business, and Science/Technology. We use the predefined training and test splits provided by the dataset and do not modify them.

Each example contains:

- A text field with the news article
- A label indicating the correct class

The dataset is suitable for this task because the texts are short and clearly related to a single topic.

3.2 Dataset Analysis

We analyzed the class distribution of the training dataset. The AG News training set is balanced, with 30,000 examples for each of the four classes, giving a total of 120,000 training examples.

Class	Number of training examples
World	30,000
Sports	30,000
Business	30,000
Science/Technology	30,000

We also printed one example from each class to verify that the labels match the content of the text. The examples clearly correspond to their assigned classes.

3.3 Model Fine-tuning

We fine-tuned the pre-trained `bert-base-uncased` model for the AG News classification task using a sequence classification head with four output labels.

The dataset was tokenized using the BERT tokenizer. We applied truncation to ensure that all inputs fit within the model's maximum sequence length.

The model was trained for three epochs on the training set. During training, the classification layer was initialized randomly, while the rest of the model parameters were loaded from the pre-trained BERT checkpoint and updated during fine-tuning.

We monitored the training and validation loss during training.

3.4 Attention Map Extraction

To extract attention maps, we enabled attention outputs in the BERT model by setting `output_attentions=True`. We then performed a forward pass on a single input sentence.

The attention output consists of attention matrices for each layer and each attention head. We selected the attention weights from the last layer and one attention head and visualized them as a heatmap.

3.5 Attention Map Comparison

We compared attention maps from the pre-trained BERT model and the fine-tuned BERT model using the same input sentence.

The resulting attention maps have the same shape in both cases. Visual inspection shows that the attention distributions differ between the two models. This indicates that fine-tuning changes how attention is distributed across tokens.

We include the attention visualizations as Figure `0.png` (pre-trained model) and Figure `1.png` (fine-tuned model).

4 Chain-of-Thought Reasoning

4.1 Model of choice

In this exercise, we went with `google/flan-t5-base` for our lightweight instruction-tuned model.

```
model = T5ForConditionalGeneration.from_pretrained("google/flan-t5-base",
    device_map="auto", torch_dtype=torch.float16)
```

4.2 Sampling 200 questions

We imported the dataset `ScienceQA` using

```
from datasets import load_dataset
...
ds = load_dataset("derek-thomas/ScienceQA", split='test')
```

From here, we defined a helper function `is_valid` which determines whether or not a question is valid, meaning that it has to have 4 choices, and no images. Afterwards, we formatted each question to match the description in the assignment. Doing so we get the following format:

```
Question: Select the living thing.
(A) cave (B) roller skate (C) kelp (D) the Great Sphinx of Giza
```

lastly, we `.dump` the formatted data into a `.json`-file, and named it `science_qa_200`.

4.3 Prompt without CoT

Now, we prompt the model. We loop through `science_qa_200.json`, where we create the prompt, and tokenize it.

```
prompt = f"{item['formatted_question']}\nAnswer:"
inputs = tokenizer(prompt, return_tensors="pt").to(device)
```

For each distinct prompt, i.e. each question in `science_qa_200.json`, we run the model on that specific prompt, and save the specific output to `outputs`. This output is then cleaned.

```
prediction = tokenizer.decode(outputs[0], skip_special_tokens=True).strip
    ()
clean_prediction = prediction.replace("(", "").replace(")", "").replace("
    .", "").strip().upper()
```

The models output is indexed from $0-3$, so in order to get an easier read, we also map those indexes to $A-D$. We then check if we got a correct answer.

```
is_correct = clean_prediction.upper().startswith(actual_letter)
```

The result of all the prompts is then stored in another `.json` file named `baseline_results.json` with the following format:

```
baseline_results.append({
    "id": i,
    "predicted": prediction,
    "clean_predicted": clean_prediction,
```

```
    "actual": actual_letter,
    "is_correct": is_correct
})
```

4.4 Prompt with CoT

Now, based on those same questions we try to use chain-of-thought reasoning with our prompt. We follow most of the structure from task 4.3, but now our prompt is

```
system_instruction = (
    """
    You are a careful reasoning assistant that answers multiple-choice
       questions about science.
    First, think step by step. Then, give the final answer. Format
       exactly as:
    Reasoning: <step 1>
    Reasoning: <step 2>
    Reasoning: <step 3>
    Answer: <final letter>
    """
)
full_prompt = f"{system_instruction}\n\nQuestion: {item['
   formatted_question']}\nAnswer:"
```

This CoT-prompt is then passed into our model, along with the 200 questions. Based on testing, the models answer may very, so we use regex pattern matching to determine the models answer for each questions.

```
find_letter = re.findall(r"(?:\(|answer is |answer: |is )?\s*([A-D])
   (?:\.|\)|$)", raw_response, re.IGNORECASE)
```

If no match is found, we write `N/A`. We again check whether or not the prediction is the same as the actual answer, and save the final output into another `.json`-file named `cot_results.json`, with the following format:

```
cot_results.append({
    "id": i,
    "question": item['formatted_question'],
    "model_response": raw_response,
    "predicted": predicted_letter,
    "actual": actual_letter,
    "is_correct": is_correct
})
```

4.5 Flip analysis

To compare the baseline prediction with the CoT prediction for each question, we make use of `pd.merge`. This allows us join the two dataframes. We then defined a helper function `categorize_flip`, which uses a truth-table-like mapping, to determine the different categories. This is saved into another date-frame, which we can analyse, and determine the accouracy of the two models, as well as the distribution of the four categories.

Metric / Category	Count	Percentage (%)
Accuracy		
Baseline Accuracy	–	49.50
CoT Accuracy	–	44.50
Outcome Distribution		
Incorrect–Incorrect	94	47.0
Correct–Correct	82	41.0
Flip-to-incorrect	17	8.5
Flip-to-correct	7	3.5
Total	200	100.0

Table 2: Accuracy and Outcome Distribution

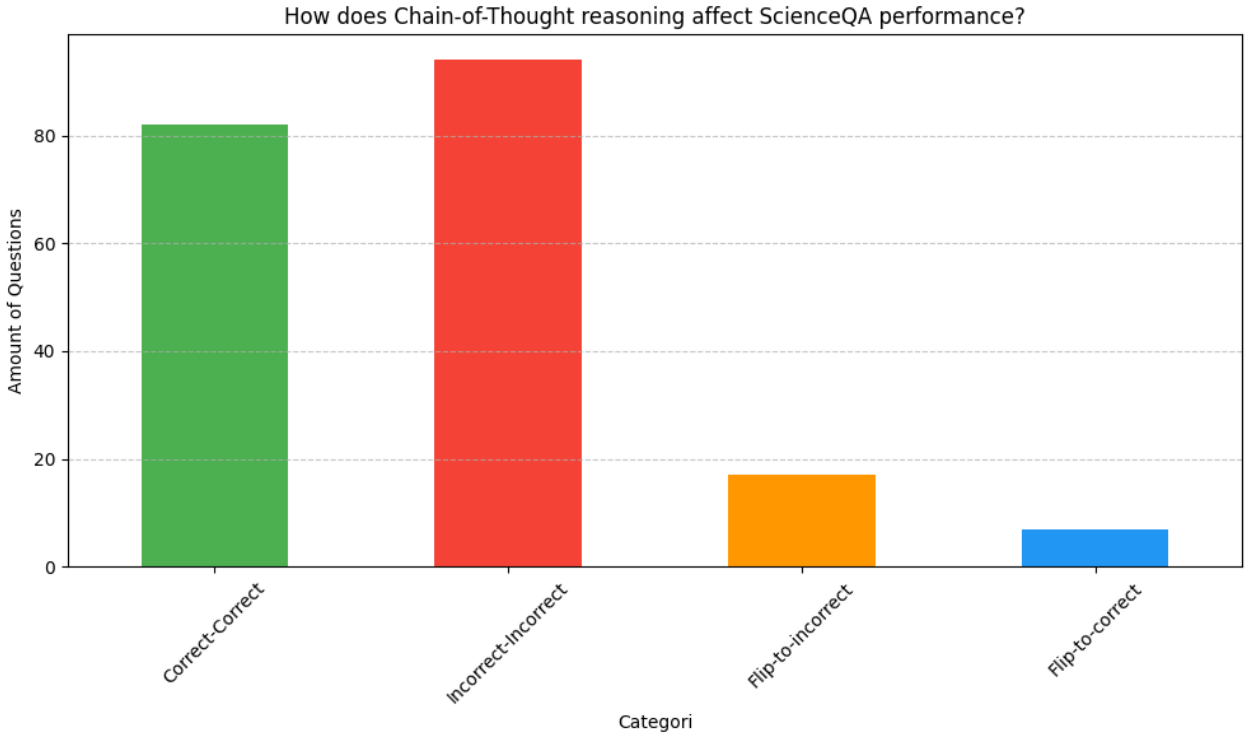


Figure 6: Bar plot comparing the distribution of the four categories.

4.6 Qualitative inspection

Flip-to-correct The baseline model often fails when it comes to simple questions regarding measurements, such as ID 4, 18, 130 and 178. The model probably sees a number, and then without considering the different units simply picks one.

The CoT model improves on this by forcing the model to formulate a verbal premise, i.e. "`A potato is 16 cm long`". When using a CoT-prompt we try to force the model to think about the correct unit, before choosing the answer.

Flip-to-incorrect For ID 188 and ID 120 we see that the reasoning is actually correct. The models writes "23 meters" or "55 millimeters", but it fails to map the correct statement with the correct answer. The reasoning is plausible, but the output mechanism fails.

For more complex logical questions such as ID 46, where it should select the solid, and ID 111, where it should pick which word is not like the others, it breaks down. The model repeats the same statement, without adding any useful insight. Her the the CoT process prevents the model from giving any meaningful answers, and it becomes nonsensical.

Incorrect-incorrect It seems there are gaps in the models knowledge. For ID 93 and ID 100, the model claims that a butterfly is a vertebra, or that moths live in the ocean. Based on these "facts" it starts reasoning, which leads to an incorrect answer.

We also see the model make up an explanation when it tries to justify an incorrect answer. For ID 180, the model claims that "A grass frog is a reptile that lives in the desert". The CoT process isn't used to find the correct answer, but instead it is used to form wrong logical reasoning for the guess, the model already have made.

Correct-Correct In the random samples extracted there is no instance of CoT providing greater insight or reasoning. The best example is when the model states that "A butterfly is a living thing. The living thing is the only living thing that can be selected.", and then correctly selects butterfly as the living thing (ID 45). There is no reasoning, and in most cases the model refuse to give reasoning, and simply state the answer.

In other cases the model gives the correct answer, but with a factually wrong reasoning. This happens in example ID 99, where the model has to select the reptile. The models reasons that "A great white shark is a reptile that lives in the ocean. The western gorilla is a reptile that lives in the ocean.", but then selects "alligator", which happens to be the correct answer.

All in all, it is not worth using CoT for this model.

4.7 Analysis & Discussion

Implementing CoT on `google/flan-t5-base` has resulted in a worsening of accouracy. The base model achieved an accuracy of 49.50%, but the CoT prompt only achieved an accuracy of 44.50%. This indicates the model didn't improve by introducing CoT. It actually made it worse. It seems that by forcing the model to think about reasoning, it essentially gets overload, and fails to provide correct reasoning, and sometimes even fails to give the correct answer, where the baseline succeeded. This can be seen by the fact that it in 17 cases, it flipped its answer from correct to incorrect, and only in 7 cases did from incorrect to correct. The models intuitive probability calculation is therefor most likely greater then its logic capability.

However, there is some cases where it seems to do better. When asked about real world metrics and units it actually seems that the models reasoning provides some help, by formulating reality into words, and avoid mixing up mm, cm and m.

On the other hand, the CoT-prompt suffers under some sort of mapping error, where it provides something the could be called reasoning, that also corresponds to the correct answer, and still give the wrong answer as a final answer. Furthermore, under more complex logical reasoning questions the model suffers under a logical collapse, where it gets stuck in a loop of non-sense or circular definitions such as ID 111.

The question of whether the model shows genuine reasoning is easy to answer. In the correct-correct category, we frequently see that the reasoning functions as an unreliable post-hoc explanation rather then a decision making process. A clear example is ID 99, where the model arrives at the correct answer but support by claiming that gorillas and great white sharks are reptiles. In my opinion this proves that the reasoning often does not dictate the answer, but is just textual filler. This can at worse undermine trust in the model's factual accuracy.

Ultimately, it can be concluded that for a model of this size, the computational costs and the risk of factual errors present greatly outweighs the benefit of sometimes being able to distinguish between units.

References

- Kirillov et al. *Segment Anything*. 2023.
- Lewis-Lim et al. *Analysing Chain-of-Thought Dynamics*. 2025.