Graduate School of Information Science and Technology

Department of Computer Science

Aizawa Laboratory

Team NTNU

Multimodal Intelligent System Design: Final Project

# Visual Question Answering With PaliGemma 2: Conditional Generation, Multi-class Classification and Natural Language Inference

*Authored by:*

Nicolai Thorer Sivesind
Student ID: 48-249720

4th February 2025

# Contents

# 1 Introduction

In the continuously evolving landscape of artificial intelligence, multi-modality has emerged as the holy grail for pushing the boundaries of what machines are capable of. Both industry and academia are focusing on foundation models that integrate and unify a range of modalities to achieve a more holistic and context-aware form of intelligence. These models represent a significant leap beyond single-domain approaches, enabling a broader understanding of real-world knowledge and unlocking a versatile range of applications.

In this project, we explore one of those applications, namely *visual multiple-choice question answering*, constructing and comparing various approaches, built upon the base architecture of the visual language model, *PaliGemma 2*:

1. *Conditonal generation with cosine similarity selection*
2. *Multi-class classification*
3. *Natural language inference using SWAG-based data decomposition*

Additionally, for the latter two approaches, we also assess two distinct pooling strategies for intermediate feature extraction from the final hidden state and the linear output layer of our non-generative approaches - attentive vs. last-token pooling.

The source code for our project can be found at github.com/Brilleslangen/Multimodal-VQA

## 1.1 Cooperation and Personal Contribution

The foundational work this report is built upon is a result of a cooperation between me, *Nicolai Thorer Sivesind* and my friend and thesis cowriter, *Erlend Rønning*. We have utilized each other's domain knowledge and distinct qualities to research and discuss how to approach the problem and what approaches to pursue.

My main responsibility has been to build the code base for our models and the training-evaluation pipeline, while my partner has focused on the problem of computational resources, model parameterization, and running the code on our lab's cluster. Throughout the entire project, we have been sitting side by side, continuously helping each other and discussing problems, bugs, and other related issues.

# 2 Related Works

This section will concisely formulate the research relevant to our methods.

## 2.1 PaliGemma 2

PaliGemma 2 is an open-source multilingual visual language model developed by Google (Steiner et al., 2024), inspired by their closed-sourced multimodal model PaLi (Chen et al., 2023), and which has been explicitly trained on Japanese data. It is a further development of its predecessor, PaliGemma (Beyer et al., 2024), and utilizes SigLIP (Zhai et al., 2023) for image encoding and Gemma 2 (Riviere et al., 2024) autoregressive decoding.

PaliGemma 2 has been built with focus on model compactness for fine-tuning and utilization for the general public. It has demonstrated exceptional performance across a wide array of vision-language tasks, establishing itself within the field of vision-text processing (*PaliGemma model README* 2025). The model it is inspired by, PaLi (Chen et al., 2023), currently holds the state-of-the-art benchmark performance on VQA 2.0 dataset at PapersWithCode with an accuracy of 83.4 % (*Papers with Code - VQA v2 test-dev Benchmark (Visual Question Answering (VQA))* 2025).

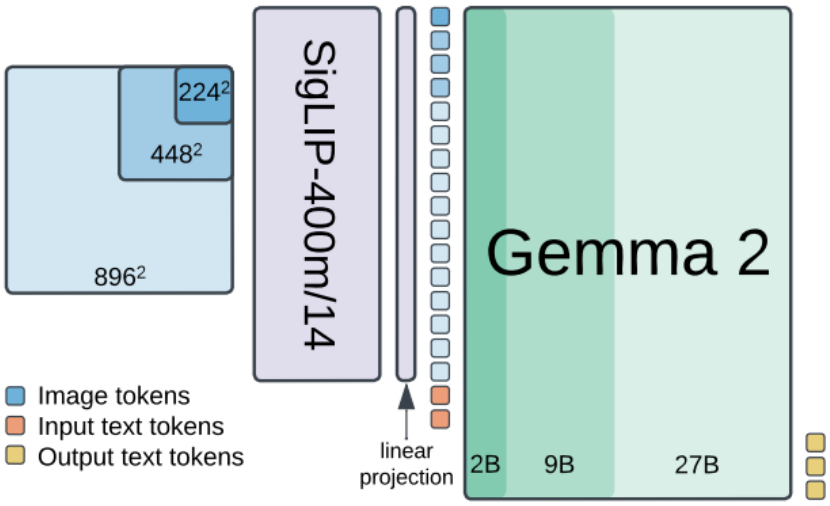
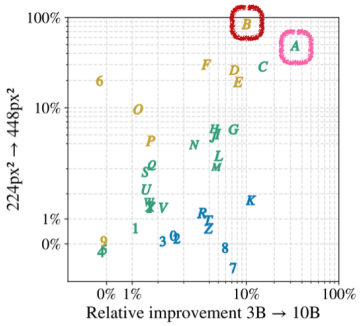



**Figure 1:** *PaliGemma 2 architecture illustration retrieved from PaliGemma 2 paper (Beyer et al., 2024). SigLIP encodes images into a latent representation that is projected into the embedding space of Gemma 2. Following this embedding translation, the two modalities are additively fused through concatenation. Gemma 2 finally decodes the resulting embedded sequence auto-regressively to produce output sequences.*

There are two main parameters that impact the performance of PaliGemma 2 - image resolution and model parameter size. This is especially true for datasets containing more abstract imagery such as *DocVQA* and *InfoVQA*, which consist of documents and info-graphics respectively. For these kinds of benchmarks, image size seems to be the biggest dictator of performance, and this might be attributed to the complexity of the images. Figure 2a shows a significant performance gain when increasing image size from $224^2$ to $448^2$, more so than parameter count, where the differences between 10B and 28B are negligent. The gains made between the image sizes are significantly greater in *DocVQA* and *InfoVQA* compared to the surrounding benchmarks (Google, 2025). From Figure 2b, it is apparent that these kinds of benchmarks with abstract, complex, or graphical imagery are the ones that gain the most from an increase in image and parameter sizes (Beyer et al., 2024). This suggests that to optimize performance in such environments, one should prioritize image size over parameter count and that a parameter count of 10B is sufficient for optimal performance.

**(a)** PaliGemma 2 Model Card Benchmarks (Google, 2025). The model variations are noted in the format `imgsize-parametercount`. For both *DocVQA* and *InfoVQA* the biggest model in the $224^2$ tier is outperformed by the smallest model in the next image size tier at $448^2$.

| Benchmark | 224-3B | 224-10B | 224-28B | 448-3B | 448-10B | 448-28B |
|---|---|---|---|---|---|---|
| CountBenchQA | 81.0 | 84.0 | 86.4 | 82.0 | 85.3 | 87.4 |
| DocVQA (val) | 39.9 | 43.9 | 44.9 | 73.6 | 76.6 | 76.1 |
| GQA | 66.2 | 67.2 | 67.3 | 68.1 | 68.3 | 68.3 |
| InfoVQA (val) | 25.2 | 33.6 | 36.4 | 37.5 | 47.8 | 46.7 |
| MARVL (avg5) | 83.5 | 89.5 | 90.6 | 82.7 | 89.1 | 89.7 |




**(b)** Performance chart showing how image sizes and model parameter count relatively impact the performance across various VQA benchmarks. *DocVQA* and *InfoVQA* are the most sensitive benchmarks (Beyer et al., 2024).

**Figure 2:** Charts showing how image size and parameter count impact performance across benchmarks. Benchmarks containing abstract and graphical imagery seem to be the most sensitive.

### 2.2 SWAG

SWAG (Zellers et al., 2018) is a natural language dataset used for evaluating language models' ability to infer the correct continuation of a sentence out of a set of four suggested options. This is done by splitting a single data point into pairs of start + ending options. Each sentence pair is passed through the model to receive a coherence score and the sentence pair that receives the highest score out of the four options, is classified as the selected option by the model. The authors describe this approach as 'grounded commonsense inference, unifying natural language inference and commonsense reasoning' (Zellers et al., 2018).

On stage, a woman takes a seat at the piano. She
a) sits on a bench as her sister plays with the doll.
b) smiles with someone as the music plays.
c) is in the crowd, watching the dancers.
**d) nervously sets her fingers on the keys.**

A girl is going across a set of monkey bars. She
a) jumps up across the monkey bars.
b) struggles onto the monkey bars to grab her head.
**c) gets to the end and stands on a wooden plank.**
d) jumps up and does a back flip.

The woman is now blow drying the dog. The dog
**a) is placed in the kennel next to a woman's feet.**
b) washes her face with the shampoo.
c) walks into frame and walks towards the dog.
d) tried to cut her face, so she is trying to do something very close to her face.

**Figure 3:** Example from the SWAG dataset retrieved from its paper *SWAG* (Zellers et al., 2018). Correct options are marked in **bold**.

## 3 Methods

To approach the *Complex Visual Data Understanding Challenge* of this project, we designed three different pipelines for comparison, all of which were built on top of the pre-trained model, *PaliGemma 2*. This section presents these approaches and related configurations.

### 3.1 Conditional Generation with Cosine Similarity

The baseline model for our testing is based on the off-the-shelf model *PaliGemmaForConditionalGeneration*, which is the only version of PaliGemma 2 that is fine-tunable out-of-the-box on Hugging Face.

Conditional Generation is the approach of training or *conditioning* a model to generate a sequence in a specific format based on the provided input. This is useful in cases where the set of possible outputs is not fixed or is very large such as in visual question answering where the answer depends on the provided context typically containing an image and question.

For our conditional generation models, hereby referred to as `COND_GEN`, we format a prompt to contain the image, question, and the corresponding options provided in the dataset and fine-tuned the model to output one of the answers. As an additional layer of assurance, we calculate the cosine similarity layer between the model output and all of the options and assign the option with the highest score as the model answer. This aims to potentially avoid issues where the output does not perfectly match any of the options. It also increases the base performance to 25%, as that is the accuracy of random guessing.

```
"<image> Who was born on 25th July 1876?
<separator>
Options: Elmer Drasher | Saveria Orazi | Giovanni Vitale | Antonio
↪ DiGianno
<separator>
Answer:"
```

**Listing 1:** Prompt formatted for our COND_GEN models. The image, question, options, and beginning of the answer sequence, which the model is to continue, were provided.

In listing 1, the prompt for COND_GEN is presented. For sequence separation, the token <separator> was used to separate the question, its options, and the answer. This token and the formatting are based on an article by Google on how to use Gemma for custom classification (*Showcasing Agile Safety Classifiers with Gemma* 2025). PaliGemma 2 utilizes the same tokenizer GemmaTokenizerFast, with extended input to handle the projected image tokens, which PaliGemma 2 uses for its text (*PaliGemma model README* 2025).

## 3.2 Multi-class Classification

In addition to COND_GEN, we have developed two custom approaches. The first is conventional multi-class classification with four classes, one for each option. This is not a common approach for multiple-choice question answering, but we were curious to see if the model is able to learn how the relative position of the correct answer maps to the desired output, given a sequence of options. This allows the model to provide an answer directly by giving the highest logit score to the corresponding output index of its answer, bypassing the need to generate text and mapping it to an option.

As there is no prebuilt PaliGemma architecture for multi-class classification, we took inspiration from the source code of PaliGemmaForConditionalGeneration (*Modelling PaliGemma* 2025), and tailored it for our classification approach. We set up the model architecture similarly to Figure 2 with the same vision tower, language model, and their respective embedders. Finally, we swapped out the generation head with a linear classification layer, which took the pooled final hidden state as input, and output 4 logits - one for each option class. This adjustment also required customization of the forward-function and adjustments to various fine-tuning pipeline methods. We refer to this model as MULTI-CLASS

```
"<image> Which competitor is participating in the 'Sword' event at
↪ 10:35?
<separator>
Options: A) Andrew Drummond | B) Jasmine Kue | C) Peter Guo | D) Jerome
↪ Kue
<separator>
Answer:"
```

**Listing 2:** Multi-class classification prompt. Follows the same principles as the prompt for COND_GEN in listing 1, but additionally introduces alphabetic option enumeration.

In listing 2, the prompt used for MULTI-CLASS is presented. To increase the likelihood of last token consistency to aid our last-token-pooling presented in section 3.4, we enumerated the options alphabetically. For some questions, the textual options/answers were in numbers. Subsequently, alphabetic enumeration seemed to be the most unambiguous enumeration approach.

## 3.3 Natural language inference using SWAG-based data decomposition

To be able to compare the utility of the MULTI-CLASS model-type with another similar but established approach for handling multiple-choice problems, we took inspiration from how the SWAG-dataset utilizes

a bundled natural language inference setup to select an option (Zellers et al., 2018).

In our second custom model-type, hereby referred to as `SWAG`, we have used the same architecture as `MULTI-CLASS`, but instead with a single logit output rather than 4, to produce an individual coherence score per option; We decompose the question and its options into separate question-option pairs and calculate a coherence score for each. Whereas the `MULTI-CLASS` models can answer a question in a single pass, our `SWAG` models must run 4 separate passes, one for each question-option pair. Similarly to our `MULTI-CLASS` model-type, this requires its own variations of the same set of functions in the model and the fine-tuning pipeline.

In our collate function, which prepares a batch immediately before the model receives it, we split each question into the aforementioned question-option pairs, resulting in bundles of 4 prompts per question, exemplified in listing 3. In the model's forward-function we unbundle these and iterate over them with one pass each, producing a single coherence score per option. Finally, these scores are reassembled into a four-logit vector that matches the output format of the `MULTI-CLASS` model. The remainder of the process, including loss calculation, remains identical for both approaches.

```
["<image> Which competitor is participating in the 'Sword' event at
 ↪  10:35?
<separator>
Hypothesis: Andrew Drummond",

"<image> Which competitor is participating in the 'Sword' event at
 ↪  10:35?
<separator>
Hypothesis: Jasmine Kue",

"<image> Which competitor is participating in the 'Sword' event at
 ↪  10:35?
<separator>
Hypothesis: Peter Guo",

"<image> Which competitor is participating in the 'Sword' event at
 ↪  10:35?
<separator>
Hypothesis: Jerome Kue"]
```

**Listing 3:** SWAG-based natural language inference prompt set. Each option is separated into its own question-option pair prompt. Instead of *'answer'* to initiate the model task, we use *'Hypothesis'*, which is a popular formulation when performing natural language inference with language models.

## 3.4 Intermediate pooling strategies

For our two custom non-generative model types, we explored different ways to pool the final hidden states. The first approach, last-token pooling, is the most commonly used in autoregressive models for classification. This means using only the hidden state that would be used to generate the last token of the model sequence, relying on this final token's hidden state, which has been sculpted by autoregressively processing each preceding token of the input sequentially. The idea is that this final hidden state captures contextual information from all preceding tokens.o

Our second pooling approach is using attentive pooling. By adding a unidirectional attention layer between the hidden states for all tokens and pooling them using single-headed attention, we allow the model to learn its own pooling strategy by focusing on the parts of the sequence most relevant for minimizing loss. However, it also introduces an additional set of weights that must be trained.

In the paper *Pooling And Attention*, these two pooling strategies, referred to as *EOS-pooling* and *Causal*

*pooling respectively*, are shown to be highly competitive for classification tasks, performing on par with more complex designs (Tang and Yang, 2024).

## 3.5 Data and General Settings

The previously described setups of this chapter result in 5 architecturally different models - each of the 3 models `COND_GEN`, `MULTI-CLASS` and `SWAG`, where the latter two utilize last-token-pooling as standard, and we have two additional architectures when instead utilizing attention based pooling. On top of these 5 model variations, we have various settings that apply to all models.

### 3.5.1 Data splits

From the project, we were given three datasets:

1. **Train**: 2000 samples
2. **Development**: 500 unlabeled samples
3. **Test**: 619 unlabeled samples

To perform a preliminary evaluation of our models to select the contender to run inference for the development and test splits, we split our training data in a 90/10 split, resulting in a training set of 1800 training samples and in-training validation samples. This lets us produce comparative performance charts to document the training efficiency and performance of each approach. For our final model, we selected the best-performing approach and trained on all 2000 training samples to maximize exposure to relevant data and optimize performance.

### 3.5.2 Low-Rank Adaptation of Large Language Models

Training large multi-modal models uses a lot of memory. We therefore utilize (LoRa), to train a compatible smaller set of adaptation weights for our model's transformer layers instead of it's actual weights. This proved to be effective in reducing computational needs significantly. As training times were extreme without, this was used for all our models.

### 3.5.3 Frozen vision tower

To further reduce training times, we froze the vision tower of the model during training. This component of the model, which essentially embeds the images and projects them into the language models embedding space, has gone through extensive pre-training through the 3-stage training process of `PaliGemma` (Beyer et al., 2024), and based on this we theorized that our deliberately sized fine-tuning procedure would not improve the vision embeddings at a sufficiently contributory level. We did one test on our best model, which confirmed that training with an unfrozen vision tower did not provide any gains in terms of performance. All our models have, therefore, been trained with a frozen vision tower.

## 3.6 Hardware and Computational Constraints

For development, we have utilized a personal MacBook Pro M2-Max with 32GB of unified memory. This enabled the training of models limited to $224^2$ image size, 3B parameters, and a maximum batch size of 1. For actual model training, we therefore utilized our laboratory's computer cluster to be able to train with bigger image sizes and parameter counts.

When running inference, we were unable to distribute our model across multiple GPUs. Consequently, while performing in-training evaluation to select the best model for retraining on all training data, we were

limited to using a single A100 80GB GPU. The same constraint applied when running inference on the development and test datasets. Additionally, as image sizes increase, the size of intermediate feature maps grows quadratically, significantly expanding the memory required for both activations and gradient storage. These constraints confined us to fine-tuning our model at an image size of $448^2$, which is not optimal. Even at a 3B parameter count, attempting $896^2$ led to out-of-memory errors after around 120 training samples, suggesting a potential memory leak. Despite extensive efforts, we could not resolve this issue, limiting us to $448^2$ images with a 10B parameter model. For our `SWAG` models, we had to further reduce the parameter count to 3B, since their 4-pass nature described in section 3.3, effectively treats a single data point as a batch size of four, exceeding our available memory.

# 4 Results and Analysis

In this section, we will concisely present and analyze the results from our 5 model variations described in section 3. The figures presented in this chapter are all with $448^2$ image sizes for the reasons stated in sections 2.1 and 3.3.

## 4.1 Training Evaluation Performances

### 4.1.1 3B Model Class

To be able to compare all our models at an equal parameter size, we have performed a 2-epoch training run for each of our 5 model variations at 3B parameters each. The results from these tests are displayed in Figure 4.

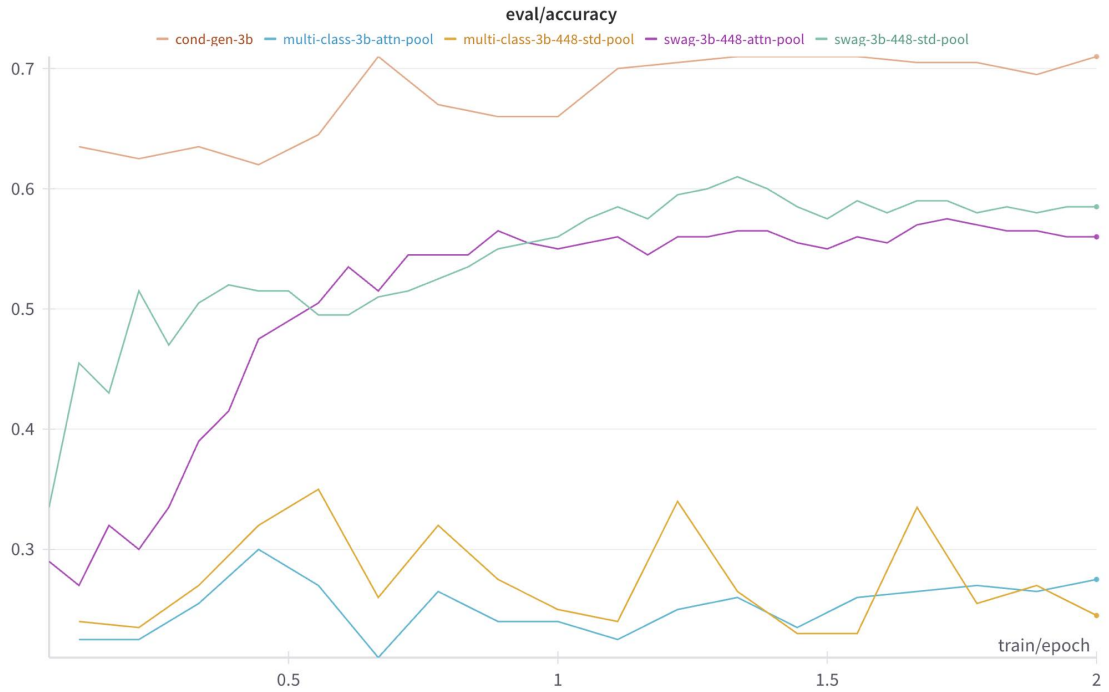



**Figure 4:** 3B model comparison chart presenting the evolution of model accuracy on 200 samples test over 2 epochs of 1800 samples each. *std-pool* refers to last-token-pooling, while *attn-pool* refers to attentive pooling.

In our 3B model comparison chart presented in Figure 4, there are three separated tiers of performance, each corresponding to a specific model architecture. At the bottom, we find the `MULTI-CLASS` model with and without attention pooling. These models fluctuate around the baseline accuracy of 25% for the entirety of their training, indicating that the model at this parameter count is unable to infer the relationship between

its four output logits and the relative position of its available options. We can conclude that this is the cause, as the other model genres all perform above baseline.

The next tier groups the results of the SWAG models. While not performing on the level of the top tier, they are able to do actual cohesive question answering, eventually approaching a stable performance of around 60% accuracy. Compared to COND_GEN, which maintains a relatively high accuracy from its first evaluation after around 70 samples, the SWAG models start closer to baseline but have a steeper learning curve. This is due to the nature of the pre-trained model they are built upon - while COND_GEN performs the task it has been natively pre-trained for, the SWAG models have been repurposed and introduces new layers with untrained weights that needs adjustment. The last-token-pooling SWAG model adds one new linear layer, while the attention-based-pooling SWAG model has two, resulting in relative learning offsets until 0.5 epochs.

The standard approach for question answering, COND_GEN, is the clear champion of the 3B parameter class, keeping a stable 10 percentage point gap to the second tier of models, and reaching a final accuracy of around 70%.

### 4.1.2 10B Model Class

Building on the takeaways from the previous section, the next models in the next parameter class of 10B are similarly evaluated. Here, the SWAG models are not included due to memory constraints described in section 3.6

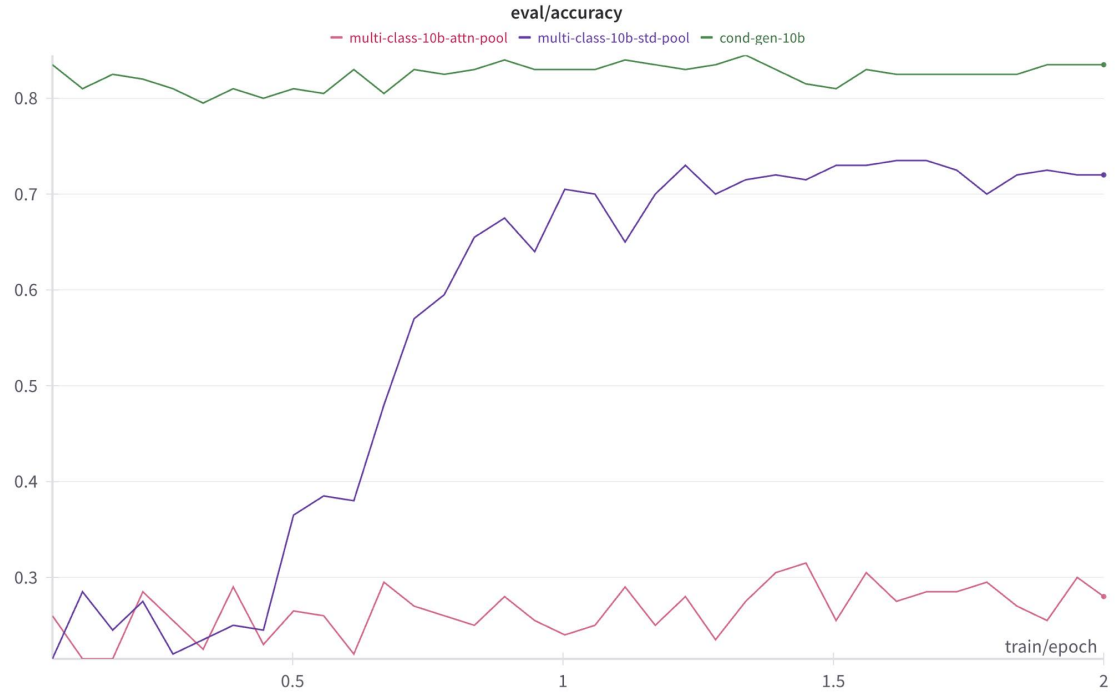



**Figure 5:** 10B model comparison chart presenting the evolution of model accuracy on 200 test samples over 2 epochs of 1800 samples each. *std-pool* refers to last-token-pooling, while *attn-pool* refers to attentive pooling.

Figure 5 showcasing the 10B in-training performances shows similar characteristics as Figure 4, having three distinct performance tiers. The most interesting difference is that MULTI-CLASS, with an increased amount of model parameters, now are able to infer the relationship between the correct option in its prompt to its output logits position. But, this only applies to MULTI-CLASS with last-token-pooling. The additional layer of attentive pooling seems to introduce too much complexity even for a 10B model. Despite learning to infer this relationship, it does not reach the same level as COND_GEN, maintaining a gap of 10 percentage points similar to the 3B parameter class evaluation.

Another difference has to do with COND_GEN and its learning curve. Compared to COND_GEN of 3B, which

starts 60% accuracy and climbs to 70% during training, 10B `COND_GEN` starts close to its peak evaluation during training, producing a negligent slope across its training cycle. Its peak accuracy around 1.4 epochs is at 84% accuracy. This may potentially suggest that the pre-training of this model is close enough to the fine-tuned task that this model at 10B is able to essentially perform optimally already in zero-shot settings.

### 4.2 Unlabeled Evaluations

This section presents the result of our best model on the unlabeled datasets that are evaluated externally. Additionally, our results can be found in the leader board as *NTNU*.

From our charts presented in our Figures 4 and 5, `COND_GEN` with 10B was the best performing model, and its base model trained on all 2000 training samples produced the level of performance presented in Figure 6.

| ID # | File name | Date | Status | Score |
|---|---|---|---|---|
| 223445 | ntnu.zip | 2025-01-30 16:27 | Finished | 0.71 |
| 222710 | ntnu.zip | 2025-01-29 15:28 | Finished | 0.69 |
| 221372 | ntnu.zip | 2025-01-27 14:56 | Finished | 0.71 |

**Figure 6:** Various submissions for the development dataset.

For both the development dataset and the test development set, our top achieved performance arrived at **71%** for both sets, which put our submission at a disappointing 31st out of 35 entries. One thing to note is that this performance is significantly lower than how our model performed during our in-training evaluations, which provides ground for discussions.

## 5 Conclusion, Discussion and Reflections

This section will conclude, discuss and reflect on some interesting aspects of our project and the results.

### 5.1 Comparing our Novel Architectures

While the standard approach of `COND_GEN` seems to offer the easiest and most proficient performances, it has been interesting to develop more novel approaches to the problem. When developing these architectures, our biggest concern was the potential of `MULTI-CLASS` to not be able to infer the positional relationships between options and output logits, which motivated our additional approach `SWAG` - a similar approach that omits the positional complexity at the cost of computation. Our results show that this was a valid concern, underlined by the 3B performances shown in Figure 4, but that the model was able to overcome this as its parameter size was increased. Despite this, our memory constraints refrained us from being able to provide a comparison at similar parameter sizes, which leaves us only with speculations; if we assume a similar performance gain as `COND_GEN` of 10% from 3B to 10B, `SWAG` would outperform the `MULTI-CLASS`, but in order to truly know we need to evaluate them in the same context.

Regarding our pooling strategies, attention-based pooling does, for this task, does not seem to offer anything else than increased complexity and additional computation. While it has been interesting to see if an autoregressive model can benefit from being able to attend various hidden states of its output sequence when performing classification, the standard strategy of last-token-pooling combined with single-letter enumeration of the options seems to offer the most simple yet effective solution, outperforming attentive-pooling in both `MULTI-CLASS` and `SWAG`, which suggests that the model encapsulates enough of the previous context within its final token hidden state, to enable cohesive inference.

As to why last-token-pooling works for `MULTI-CLASS` and attentive pooling does not; The letter enumeration of options in the prompt may provide the model a way to answer the question by producing the hidden state

that in generative mode would have mapped to the letter of its answer, enabling the classification layer a conceptually simple way to map each letter representation to an output logit. Adding a wall of untrained weights between that, namely the attentive-pooling layer, may simply introduce unnecessary complexity beyond the capabilities of the model. This would be interesting to look at, along with the saliency map of the attention-pooling layer, to see which sections of its hidden states it uses, if it does not simply converge towards an indirect last-token-pooling. But, as our focus was firmly on the issue of the performance discrepancy between our in-training evaluation performance and the external evaluation performance, this was down-prioritized.

## 5.2 Conditional Generation and Evaluation Discrepancies

The clear winner of our training evaluations was `COND_GEN`, showcasing evaluation performances above 80%. We therefore expected to see similar performance when we sent in our initial submission of the development dataset, but to our surprise, it showcased a much lower performance of 71%, and despite extensive efforts, we were unable to find out why and potentially fix it.

One possible cause of the discrepancy is our inference pipeline. How we perform evaluation during training differs slightly from how we do it during inference. In training, the model logits for all sequence positions are provided by the Hugging Face trainer pipeline to its compute_metrics function, where one can define custom evaluation methods. Here, we had to perform argmax and decoding to text before clipping out the last sequence marked by its end-of-sequence token. In comparison, when reloading our model for inference, despite initially attempting to replicate this approach, our only functional workaround was using the model's generate function, which directly produced a textual output where we only had to extract its final sequence. To investigate potential differences here, we ran our model on 200 samples from our training dataset, which gave us an accuracy of 97%, confirming that such discrepancies were not the cause. Now, 97% is very high, and may potentially indicate overfitting to its training data, but as it has been trained on this data twice, it is hard to say.

Another issue could be with how we produce the CSV files, as the leader board submission seemed to require the file_name column to arrive in the exact same order, while Hugging Face Datasets inherently processes datasets containing images alphabetically, we had to remap the answers using the original metadata-file. This may potentially have introduced mismappings. While it is possible, it is unlikely considering that it is unusual for this type of mapping to work for most and not all entries, as we are only lacking 10 percentage points from our training performances.

Assuming the aforementioned issues were not the cause we have attempted various fixes:

- Trained a quantized 28B model, which performed around the 50% mark. Showing bigger, but quantized was not a solution.
- Trained a model without freezing the vision components. No impact on performance.
- Added drop-out in our LoRa config to reduce potential overfitting. Increased performance by 2 percentage points.
- A lot of sanity tests in various steps in the pipeline of the models to ensure everything did what it should.

Despite these efforts, we were not able to increase our model performances, which leaves us with three potential causes for the low performance:

1. *LoRa*: Deactivating LoRa increased the training times extensively, which became slower and slower as it went on. We therefore did not get to produce a model without LoRa. Our research into its potential to make such a negative impact on conditioning tasks seemed unlikely (Jin et al., 2024).
2. *Representative test set*: Our custom test set used during training consisted of 200 samples from the training data, while the unlabeled datasets held 500+ entries. Our 200-piece test set could be less representative of the general dataset distribution, potentially providing us with an easier time during training evaluation.

3. *Image Size*: The most probable solution to increasing our general performance would be to train a model with an image size of 896, which we were unable to do.

We are not quite sure what the cause of the discrepancy is, but our actual performance of 71% may simply be due to hitting the roof of the model. This theory aligns with the similar benchmarks provided in the PaliGemma 2 model card, peaking at 46.8% and 76.6% for the *InfoVQA* and *DocVQA* respectively (Google, 2025).

# 6 Final Remarks

For the past month, me and my co-writer Erlend Rønning, have been investing weeks into this project, conducting novel research we found interesting, while also attempting to perform well on the challenge leader board. Despite our failure in the latter we still gained a lot of experience field of multimodality, which will be very useful in our upcoming master's thesis where we will build multimodal systems for audio transformation, replicating the cocktail-party effect.

Thank you for an interesting course, with valuable insights into the current landscape of artificial intelligence.

# Bibliography


Steiner, Andreas et al. (Dec. 2024). *PaliGemma 2: A Family of Versatile VLMs for Transfer*. arXiv:2412.03555 [cs]. DOI: 10.48550/arXiv.2412.03555. URL: http://arxiv.org/abs/2412.03555 (visited on 25th Jan. 2025).

Chen, Xi et al. (June 2023). *PaLI: A Jointly-Scaled Multilingual Language-Image Model*. arXiv:2209.06794 [cs] version: 4. DOI: 10.48550/arXiv.2209.06794. URL: http://arxiv.org/abs/2209.06794 (visited on 25th Jan. 2025).

Beyer, Lucas et al. (Oct. 2024). *PaliGemma: A versatile 3B VLM for transfer*. arXiv:2407.07726 [cs]. DOI: 10.48550/arXiv.2407.07726. URL: http://arxiv.org/abs/2407.07726 (visited on 25th Jan. 2025).

Zhai, Xiaohua et al. (Sept. 2023). *Sigmoid Loss for Language Image Pre-Training*. arXiv:2303.15343 [cs]. DOI: 10.48550/arXiv.2303.15343. URL: http://arxiv.org/abs/2303.15343 (visited on 25th Jan. 2025).

Riviere, Morgane et al. (Oct. 2024). *Gemma 2: Improving Open Language Models at a Practical Size*. arXiv:2408.00118 [cs]. DOI: 10.48550/arXiv.2408.00118. URL: http://arxiv.org/abs/2408.00118 (visited on 25th Jan. 2025).

*PaliGemma model README* (2025). en. URL: https://github.com/google-research/big_vision/blob/main/big_vision/configs/proj/paligemma/README.md (visited on 3rd Feb. 2025).

*Papers with Code - VQA v2 test-dev Benchmark (Visual Question Answering (VQA))* (2025). en. URL: https://paperswithcode.com/sota/visual-question-answering-on-vqa-v2-test-dev (visited on 25th Jan. 2025).

Google (2025). *PaliGemma 2 model card*. en. URL: https://ai.google.dev/gemma/docs/paligemma/model-card-2 (visited on 25th Jan. 2025).

Zellers, Rowan et al. (Aug. 2018). *SWAG: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference*. arXiv:1808.05326 [cs]. DOI: 10.48550/arXiv.1808.05326. URL: http://arxiv.org/abs/1808.05326 (visited on 3rd Feb. 2025).

*Showcasing Agile Safety Classifiers with Gemma* (2025). en. URL: https://ai.google.dev/gemma/docs/agile_classifiers (visited on 3rd Feb. 2025).

*Modelling PaliGemma* (2025). en. URL: https://github.com/huggingface/transformers/blob/main/src/transformers/models/paligemma/modeling_paligemma.py (visited on 3rd Feb. 2025).

Tang, Yixuan and Yi Yang (Sept. 2024). *Pooling And Attention: What Are Effective Designs For LLM-Based Embedding Models?* arXiv:2409.02727 [cs] version: 2. DOI: 10.48550/arXiv.2409.02727. URL: http://arxiv.org/abs/2409.02727 (visited on 4th Feb. 2025).

Jin, Xiaolong et al. (Aug. 2024). *Conditional LoRA Parameter Generation*. arXiv:2408.01415 [cs]. DOI: 10.48550/arXiv.2408.01415. URL: http://arxiv.org/abs/2408.01415 (visited on 4th Feb. 2025).