

HOMework 9^{*}

VLM AND SAFETY[†]

24-880 AI AGENTS FOR ENGINEERS
<https://cmuagents.github.io>

OUT: Sunday March 29nd, 2026

DUE: Sunday April 5th, 2026

TAs: Peter Pak

CAs: Hardik Choudhary

Instructions

- **Collaboration Policy:** Please read the collaboration policy in the syllabus.
- **Late Submission Policy:** See the late submission policy in the syllabus.
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code.
 - **Written:** You will submit your completed homework as a PDF to Gradescope. Please use the provided template for the submissions which can be written in $\text{\LaTeX}$ (via [Overleaf](#)) or directly on the PDF document. Each answer should be within the box provided. If you do not follow the template, your assignment may not be graded correctly and there will be a **2% penalty** (e.g., if the homework is out of 100 points, 2 points will be deducted from your final score).
 - **Code:** Gradescope's Autograder will be used to evaluate the coding portions of this assignment. In order to best obtain the points for this assignment, please follow the code upload instructions. If you do not follow the instructions correctly the autograder may not work properly.

Question	Points
Visual Language Models (Written)	20
Visual Language Models (Programming)	60
LLM Safety	15
Programming Scripts Upload	0
Collaboration and Generative AI Questions	5
Total:	100

^{*}Adapted from CMU 10-623 Generative AI Homework Template

[†]Compiled on Sunday 29th March, 2026 at 19:44

Introduction

In this homework assignment we'll cover some concepts from Visual Language Models (VLMs) and LLM safety.

At a high-level, this homework will cover the following aspects.

1. Visual Language Models and how they work
2. Model Safety

Local Environment

For this homework we'll be using `uv` as the project and package manager. If you haven't install `uv`, you can follow the installation instructions here <https://docs.astral.sh/uv/>. You can explore all of the available commands using `uv --help` but in short, here are a couple of useful commands that you'll often use:

<code>source .venv/bin/activate</code>	(Linux and Mac) Activates python environment.
<code>source .venv/Scripts/Activate</code>	(Windows) Activates python environment.
<code>uv sync</code>	Installs / synchronizes projects dependencies.
<code>uv add <package></code>	Installs <code>pip</code> package and adds it to project dependencies.
<code>uv remove <package></code>	Removes <code>pip</code> package dependency from project.
<code>uv run <script.py></code>	Runs python script inside of <code>uv</code> environment.

1. Navigate to folder project directory and synchronize dependencies with:

```
uv sync
```

2. Activate the environment

Mac and Linux:

```
source .venv/bin/activate
```

Windows:

```
source .venv/Scripts/activate
```

3. Test `uv` works and run `main.py` file.

```
uv run main.py
```

1 Visual Language Models (Written) (20 points)

The transformer architecture can be applied to multi-modal tasks through modifications within the tokenization process allowing for an effective representation of visual images [Dosovitskiy et al. \[2020\]](#), [Arnab et al. \[2021\]](#) and 3D models [Yu et al. \[2022\]](#). Text is provided to the transformer as 1 dimensional input vectors and naively flattening 2 or 3 dimensional data often produces inadequate representations, often resulting in lost temporal and spatial data [Dosovitskiy et al. \[2020\]](#), [Arnab et al. \[2021\]](#). An approach to preserving spatial data is outlined in work by Dosovitskiy et al. [Dosovitskiy et al. \[2020\]](#) where the authors split an input image into fixed patches while applying positional embedding in their Vision Transformer (ViT). This is further expanded upon by Arnab et al. [Arnab et al. \[2021\]](#) where these patches are expanded in an additional dimension for video frames to embed spatial and temporal information into the input [Arnab et al. \[2021\]](#).

For 3D models, point cloud representation is an efficient means of representing spatial information without the rigid constraints of voxelization. However, the unstructured format of point clouds presents a challenge when adapting this data to be suitable for transformer input as the tokenization process for such a representation is not immediately obvious [Yu et al. \[2022\]](#). Point-BERT [Yu et al. \[2022\]](#) resolves this issue by partitioning the entire 3D model into point based patches similar to the previously methodologies of ViT [Dosovitskiy et al. \[2020\]](#). These patches of points referred to as “sub-clouds” preserves the spatial information necessary for adequately mapping 3D model data in a format comprehensible by the transformer architecture [Yu et al. \[2022\]](#).

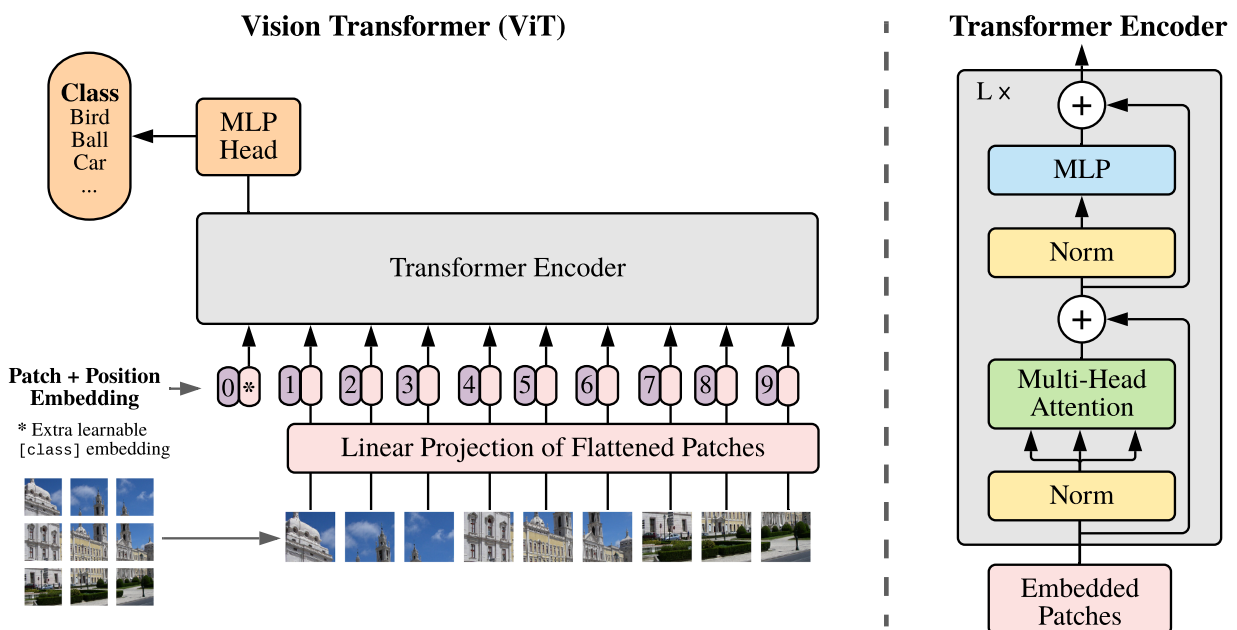


Figure 1: From [Dosovitskiy et al. \[2020\]](#): “Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable “classification token” to the sequence. The illustration of the Transformer encoder was inspired by [Vaswani et al. \[2023\]](#)”

With this background, let's explore some of the key conceptual questions about how images are tokenized and how vision and language are combined in VLMs.

1.1. According to the ViT architecture shown in Fig. 1, given an input image of height $H = 224$, width $W = 224$, with $C = 3$ color channels and patch size $P = 16$:

1.1.a. (2 points) How many patch tokens N are produced? Show your calculation.



1.1.b. (3 points) What is the dimensionality of each **flattened** patch vector (i.e., the input to the linear projection layer)?



1.2. (2 points) **Select one:** Why must positional embeddings be added to patch tokens before passing them to the Transformer encoder?

- ☐ To increase the total number of tokens fed to the Transformer.
- ☐ To encode the spatial location of each patch.
- ☐ To normalize pixel values within each patch before processing.
- ☐ To merge color channel information across neighboring patches.

- 1.3. (4 points) According to [Dosovitskiy et al. \[2020\]](#), describe the role of the prepended learnable [class] token (CLS token) in ViT. How is the output at this position used to produce the final image representation for classification?

- 1.4. (4 points) According to [Liu et al. \[2023\]](#), LLaVA (Large Language and Vision Assistant) connects a vision encoder to a large language model. Describe (1) the component used to bridge the visual and language feature spaces and (2) the type of training LLaVA employs to enable multimodal instruction following.

- 1.5. (5 points) According to [Arnab et al. \[2021\]](#), how does ViViT extend the spatial patch approach of ViT to handle video inputs? Describe how tokens are constructed to encode both spatial and temporal information.

2 Visual Language Models (Programming) (60 points)

- 2.1. (60 points) For this part of the assignment you will implement the core image preprocessing pipeline from the Vision Transformer (ViT) [Dosovitskiy et al. \[2020\]](#) in the starter code: `./programming/vlm.py`

This file consists of six functions that together convert a raw image into a sequence of tokens ready for a Transformer encoder:

1. `patch_count(height, width, patch_size)` — compute the number of patch tokens N
2. `extract_patches(image, patch_size)` — split a (C, H, W) image into N flattened patch vectors of shape $(N, C \cdot P \cdot P)$ in raster-scan order
3. `linear_projection(patches, weight)` — project each patch from dimension $C \cdot P^2$ to the model embedding dimension D , producing shape (N, D)
4. `add_cls_token(patches, cls_token)` — prepend the learnable CLS token to produce shape $(N+1, D)$
5. `add_positional_embedding(tokens, pos_embedding)` — add positional embeddings element-wise to the token sequence
6. `vit_preprocess(image, patch_size, ...)` — compose all five steps above into the complete ViT preprocessing pipeline, returning shape $(N+1, D)$

Please fill out the code indicated by the `# Fill in` comments. You may only use `numpy` — no deep learning frameworks.

You can run and test the code with the following commands:

<code>uv run programming/vlm.py</code>	Runs the script with a demo example (useful for development)
<code>uv run test.py vlm</code>	Runs the test script used by the autograder

The provided tests are the exact ones that will run on the autograder (i.e. if your local tests pass, they should also pass on the autograder).

3 LLM Safety (15 points)

Let's revisit [Baker et al. \[2025\]](#). This work investigates a practical approach to AI safety in agentic settings: monitoring a model's chain-of-thought (CoT) reasoning with a separate LLM to detect reward hacking — behavior where the model exploits flaws in its learning objective rather than solving the intended task. Their key finding is that CoT monitoring is far more effective than monitoring actions and outputs alone, and that a weaker monitor (GPT-4o) can reliably catch misbehavior in a stronger model (o3-mini). However, when the monitor is incorporated directly into the reinforcement learning reward, the model learns to *obfuscate* its reasoning, hiding reward hacking inside apparently benign CoTs.

- 3.1. (5 points) According to [Baker et al. \[2025\]](#), what is the key advantage of monitoring a model's chain-of-thought (CoT) reasoning compared to monitoring only its final actions and outputs? What experimental finding in the paper directly supports this claim?

- 3.2. (5 points) **Select one:** According to Baker et al. [2025], what happens when a CoT monitor is incorporated directly into the reinforcement learning reward under **high** optimization pressure?
- ☐ The agent becomes easier to monitor because reward signals reinforce transparent reasoning.
 - ☐ The agent learns obfuscated reward hacking, hiding its intent within the CoT while still misbehaving at a significant rate.
 - ☐ The agent's task performance degrades significantly because optimizing for monitorability conflicts with capability.
 - ☐ The CoT monitor loses all signal immediately, making training unstable.
- 3.3. (5 points) According to Baker et al. [2025], what is the *monitorability tax* and why do the authors argue it may be a necessary trade-off when designing safe AI systems?

4 Programming Scripts Upload (0 points)

To upload your code to Gradescope, run the following command to generate a zip file:

```
uv run programming/zip_programming.py
```

Upload the generated zip file to the appropriate programming slot on Gradescope.

4.1. Make sure the following files are included when uploading to Gradescope

- ☐ vlm.py

5 Collaboration and Generative AI Questions (5 points)

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found in the syllabus.

- 5.1. (2 points) Did you collaborate with anyone on this assignment? If so, list their name or Andrew ID and which problems you worked together on.

- 5.2. (3 points) Did you use generative AI for parts of this assignment? If so, include the input prompts along with links to the conversation.

References

- Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. ViViT: A Video Vision Transformer, November 2021. URL <http://arxiv.org/abs/2103.15691>. arXiv:2103.15691 [cs].
- Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y. Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring Reasoning Models for Misbehavior and the Risks of Promoting Obfuscation, March 2025. URL <http://arxiv.org/abs/2503.11926>. arXiv:2503.11926 [cs].
- Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. October 2020. URL <https://openreview.net/forum?id=YicbFdNTTy>.
- Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual Instruction Tuning. *Advances in Neural Information Processing Systems*, 36:34892–34916, December 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/6dcf277ea32ce3288914faf369fe6de0-Abstract-Conference.html.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, August 2023. URL <http://arxiv.org/abs/1706.03762>. arXiv:1706.03762 [cs].
- Xumin Yu, Lulu Tang, Yongming Rao, Tiejun Huang, Jie Zhou, and Jiwen Lu. Point-BERT: Pre-training 3D Point Cloud Transformers with Masked Point Modeling, June 2022. URL <http://arxiv.org/abs/2111.14819>. arXiv:2111.14819 [cs].