

HOMework 6^{*}

AGENTIC MANUFACTURING[†]

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

OUT: Sunday, March 8th 2026
DUE: Sunday, March 15th 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 L^AT_EX (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
Vision Transformer (Written)	10
Vision Transformer (Programming)	20
LLM-3D Print (Written)	65
Programming Scripts Upload	0
Collaboration and Generative AI Questions	5
Total:	100

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

[†]Compiled on Sunday 8th March, 2026 at 22:35

Introduction

In this homework assignment we'll look into the different aspects of LLM-3D print

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

1. How a large language model is able to interpret images.
2. Basic implementation of underlying concepts in ViT.
3. Explore how LLM-3D Print works.

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 Vision Transformer (Written) (10 points)

Before we go into LLM-3D print by [Jadhav et al. \[2025\]](#), let's take a look into how a large language model can understand images, more specifically, the structure of the Vision Transformer. 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. An approach to preserving spatial data is outlined in work by [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. \[2021\]](#) where these patches are expanded in an additional dimension for video frames to embed spatial and temporal information into the input.

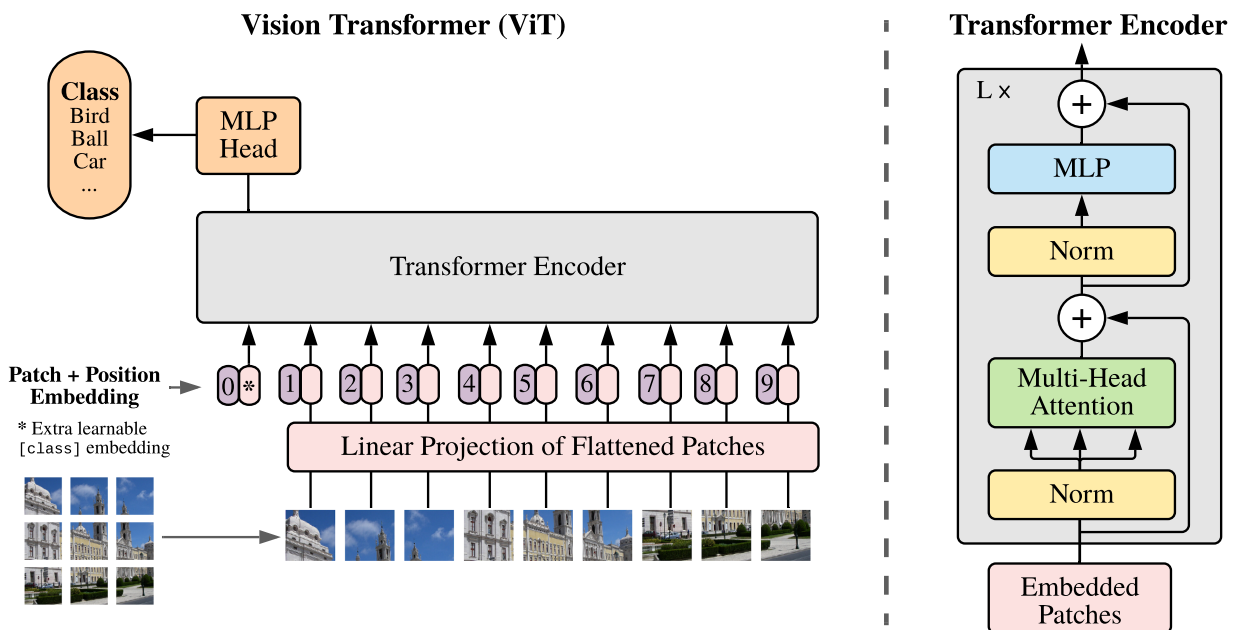


Figure 1: From [Dosovitskiy et al. \[2020\]](#), “Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach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

1.1. (3 points) The vision transformer architecture outlined in [Dosovitskiy et al. \[2020\]](#) is:

- ☐ encoder only
- ☐ decoder only
- ☐ encoder-decoder

1.2. (2 points) The tokenized input into vision transformer is:

- ☐ 1D
- ☐ 2D
- ☐ 3D

1.3. (5 points) Why can't we just provide a flattened input of an image or video into the model?

2 Vision Transformer (Programming) (20 points)

Now that we understand the ViT architecture, implement the patch embedding pipeline from [Dosovitskiy et al. \[2020\]](#) in `programming/vit.py`. There are five functions to complete. Run the tests locally with:

```
uv run python -m pytest tests/test_vit.py -v
```

- 2.1. (4 points) Implement `get_patches(image, patch_size)` that splits a numpy image array of shape (H, W) (grayscale) or (H, W, C) (RGB) into non-overlapping square patches and returns them as an array of shape $(\text{num_patches}, \text{patch_size}^2 \times C)$, ordered left-to-right, top-to-bottom.
- 2.2. (4 points) Implement `linear_projection(patches, embedding_dim, seed=42)` that applies a `torch.nn.Linear` layer (no bias) seeded with `seed` to project each patch vector to `embedding_dim` dimensions, returning a `torch.Tensor` of shape $(\text{num_patches}, \text{embedding_dim})$.
- 2.3. (4 points) Implement `positional_encoding(num_patches, embedding_dim)` that produces sinusoidal positional encodings following the formulation from *Attention is All You Need* [Vaswani et al. \[2023\]](#), returning a `torch.Tensor` of shape $(\text{num_patches}, \text{embedding_dim})$ where every row is unique.
- 2.4. (4 points) Implement `embed_image(image, patch_size, embedding_dim, seed=42)` that ties the above together: extract patches, apply linear projection, add positional encodings, and return a tensor of shape $(\text{num_patches}, \text{embedding_dim})$.
- 2.5. (4 points) Implement `embed_text(text, embedding_dim, seed=42)` that embeds a text string using character-level `torch.nn.Embedding`, averages over all characters, and returns a `torch.Tensor` of shape $(1, \text{embedding_dim})$, giving text and image patches a shared feature space.

3 LLM-3D Print (Written) (65 points)

LLM-3D print by Jadhav et al. [2025] investigates the application of a large language model (variants of ChatGPT 4 by OpenAI) enabled agentic system for optimizing fused deposition modeling printing parameters. By leveraging the vision capabilities of ChatGPT 4, optical images taken of the build layer are evaluated by the LLM for any potential defects and the necessary actions are executed to address these issues. The agentic system utilizes the LangChain library as its primary framework where in which the supervisor agent orchestrates tasks to its subagents for tasks such as planning, information extraction, and solution execution. These actions significantly improved the print quality of tested parts, addressing potential defects in near real-time, further validated by compression tests with increased peak load sustained by LLM-3D print fabricated parts compared to parts build without the assistance of the agentic system.

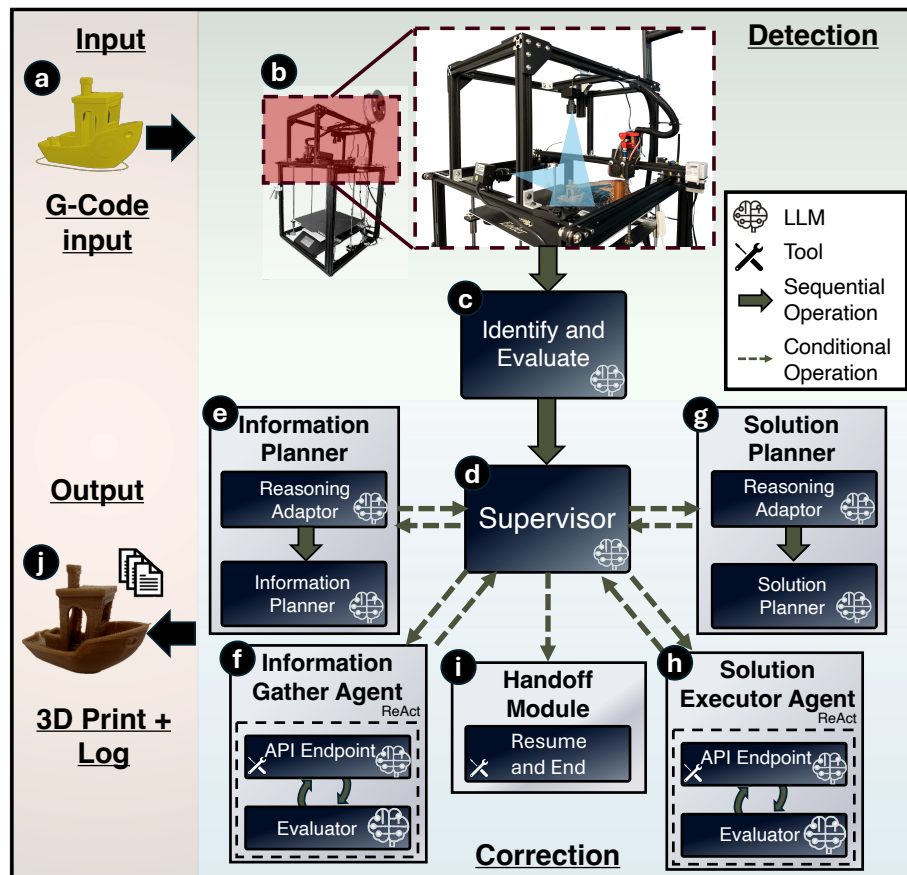


Figure 2: From Jadhav et al. [2025] “Schematic of the Proposed Framework. (a) The process begins with uploading a part’s G-code file to the 3D printer. (b) The printer is equipped with two frame-mounted cameras. (c) After each layer is printed, the extruder moves to the home position, and two images of the current print state are captured. These images are analyzed by the LLM, which evaluates the print, makes observations, and identifies any failures. (d, e) If failures are detected, the LLM supervisor invokes the information planner. (f) The executor then carries out the information gathering plan. (g) Afterwards, the solution planner is activated by the supervisor. (h) The solution plan is executed by another executor. (i, j) Loop ends with the supervisor invoking the handoff module to resume the print.”

In this section, you will follow along with a local open-source re-implementation of the LLM-3D Print pipeline using Qwen2-VL-2B-Instruct; run it with:

```
uv run python programming/llm_3d_print.py
```

- 3.1. (5 points) How does the agentic system used by LLM-3D print coordinate its agents? (Reference: `get_graph()` in `llm_3d_print.py`)
- ☐ A central agent dynamically routes tasks to specialized worker agents based on the current state
 - ☐ Each agent passes its output to the next agent in a fixed, predetermined sequence
 - ☐ Agents independently read from and write to a shared memory space with no central coordinator
 - ☐ Agents trigger each other by monitoring and responding to events in a shared event queue
- 3.2. (5 points) In the LLM-3D Print pipeline, GPT-4 Vision analyzes each print layer image and returns a response in a fixed structured format. Which of the following best describes the required output format? (Reference: `image_system_prompt` in `llm_3d_print.py`)
- ☐ A single pass/fail label with a numeric confidence score
 - ☐ A structured report containing problems, observations, failure modes, previous solution status, and print rating
 - ☐ A list of G-code commands to immediately correct the detected issue
 - ☐ Raw pixel-level bounding boxes annotating defect regions
- 3.3. (5 points) The `info_reasoning_adapter` and `solution_reasoning_adapter` functions take the generic `reasoning_planner` and `solution_reasoning` modules as input. What is their primary purpose?
- ☐ To compress the reasoning text to reduce OpenAI API token costs
 - ☐ To select, reorder, and rephrase generic reasoning steps so they focus on the specific failure type observed in the current layer
 - ☐ To translate the reasoning modules from English into G-code syntax that the printer firmware can execute directly
 - ☐ To cache common reasoning patterns so that repeated failure types do not require additional API calls

- 3.4. (10 points) Describe the role of each of the five agents in the LLM-3D Print system (`info_planner`, `recon_executor`, `sol_planner`, `sol_executor`, and `supervisor`). Explain how information flows through the system during a single failure correction cycle, from initial image analysis to the final parameter adjustment and print resumption.

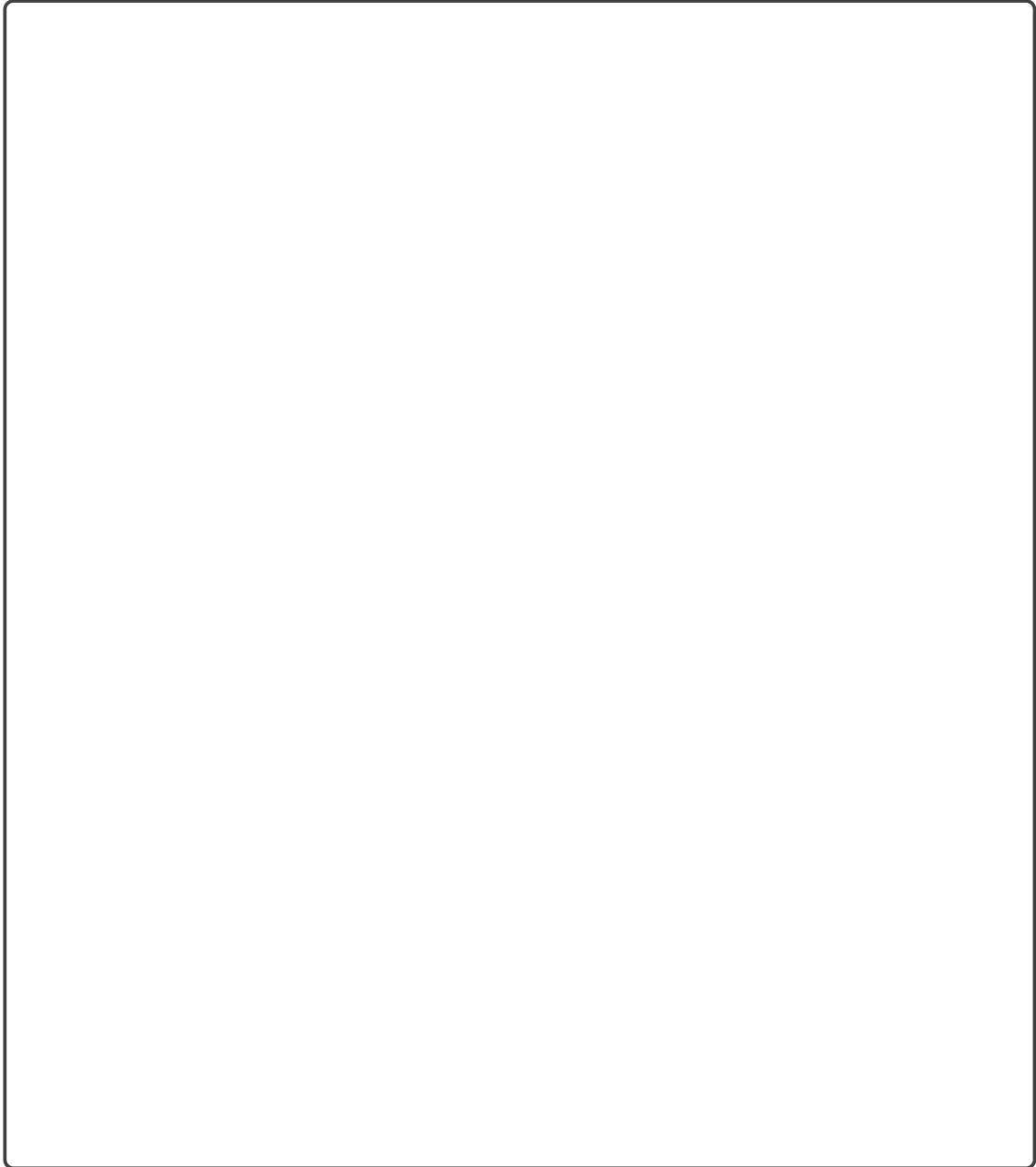
- 3.5. (10 points) The local `Qwen2-VL-2B-Instruct` model used in this implementation is a small 2-billion-parameter vision-language model. Run the script on your own 3D print layer images and observe its defect detection output. You will likely notice that it struggles to reliably identify subtle defects such as warping or stringing.

Propose and justify an alternative model — either a larger open-source vision-language model that can run locally, or a cloud API (e.g. `OPENAI_API_KEY` or `ANTHROPIC_API_KEY`, which the script already supports). For your chosen alternative: (1) identify the model by name, (2) explain why it should perform better at visual defect detection, and (3) describe any practical trade-offs compared to the current 2B local model (e.g. VRAM, cost, latency, privacy).

- 3.6. (10 points) Let's take a look at some of the other examples of defects that could occur during the FDM printing process. For this there is a nice dataset by [Hu et al. \[2024\]](#) listing 5 different defects of Cracking, Layer Shift, Off Platform, Stringing, and Warping. These images are taken with the front camera of the printer and have a couple hundred images for each. Play around with some of these images and see how well each are detected within the pipeline and list your finding below.

<https://www.kaggle.com/datasets/wengmhu/fdm-3d-printing-defect-dataset>

You'll need to copy and paste over a couple of images from each category into `data/layer_images`.



- 3.7. (10 points) The `recon_node` in the provided code loads pre-recorded printer data from a JSON file rather than querying a live printer. If LLM-3D Print were deployed on an actual 3D printer running the Moonraker API, what would need to change in the implementation? Describe at least three specific technical challenges that arise when interfacing with a live printer API compared to using pre-recorded data.

- 3.8. (10 points) Based on your analysis of the LLM-3D Print system, identify and explain at least three significant limitations of the current implementation. For each limitation, propose a concrete, technically specific improvement that would make the system more reliable, accurate, or efficient in a real manufacturing setting.

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

- vit.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].
- 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>.
- WenJing Hu, Chang Chen, Shaohui Su, Jian Zhang, and An Zhu. Real-time defect detection for FFF 3D printing using lightweight model deployment. *The International Journal of Advanced Manufacturing Technology*, 134(9):4871–4885, October 2024. ISSN 1433-3015. doi: 10.1007/s00170-024-14452-4. URL <https://doi.org/10.1007/s00170-024-14452-4>.
- Yayati Jadhav, Peter Pak, and Amir Barati Farimani. LLM-3D print: Large language models to monitor and control 3D printing. *Additive Manufacturing*, page 105027, November 2025. ISSN 2214-8604. doi: 10.1016/j.addma.2025.105027. URL <https://www.sciencedirect.com/science/article/pii/S2214860425003926>.
- 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].