Weekly Progress Report
Comparative Analysis of Transformer Architectures for Point Set Generation

Hatem Feckry (120220264) Seifeldeen Mohamed Galal (120220252)
Abdelrahman Ashraf (120220292) Ahmed Elsherbeny (120220271)

April 24, 2025

1. Introduction

This week's work has centered on two complementary thrusts: (1) pushing our convex-hexagon–detection dataset generation pipeline to higher throughput via parallelization (first on CPU threads and then by porting critical kernels to CUDA), and (2) deepening our understanding of two candidate Transformer architectures for the eventual learning task, so as to choose the right model for each subtask in our pipeline.

In addition, we perform a comparative analysis of several transformer-based architectures aimed at extending point generation capability from 25 to 32 points, under strict geometric constraints of non-colinearity and convexity, processing a scaled image input of resolution 1000×1000.

2. Conventional Dataset Generation

Our baseline generator produces random planar point sets of fixed size (16 points per set), then applies the Graham-scan algorithm to test whether a convex hexagon exists as a subset of points. Concretely, for each sample:

1. Draw 16 points independently (e.g. uniform in $[0,1]^2$).
2. Run Graham scan in $O(n \log n)$ to sort by polar angle and compute the convex hull.
3. If $|\text{hull}| \geq 6$, check all $\binom{|\text{hull}|}{6}$ subsets for convexity; otherwise reject.
4. Retain only those sets *without* any convex-hexagon subset.

Although this approach is conceptually straightforward, it becomes computationally intensive when millions of samples are required. The main bottleneck is that for the many small hulls produced, the per-sample runtime is dominated by hull construction plus subset enumeration.

3. Problems with the CUDA Prototype

While the theoretical parallelism of GPUs promised dramatic speedups, our first prototype exposed two key issues:

3.1 Thread–Launch and Data–Transfer Overhead

Each sample requires only $O(n \log n)$ operations with $n = 16$ and at most a few dozen cross-products. On GPU, the PCIe-mediated copy of input/output buffers plus kernel-launch latency (˜20 µs per launch) dwarf the actual compute (˜1–2 µs per hull). If we batch B samples per kernel, the amortized overhead per sample becomes

$$t_{\text{per sample}} = \frac{t_{\text{launch}}}{B} + t_{\text{copy}} + t_{\text{compute}},$$

where $t_{\text{compute}} \approx c\, n \log n$. Since $c \cdot (16 \log 16) \approx 2$ µs and 10 µs + 20 µs$/B \gg 2$ $sunlessB \geq 10^3$, the overheads dominate.

3.2 Serial Pipeline Bottleneck

Our initial design ran $CPU \rightarrow GPU \rightarrow CPU$: the CPU would generate random seeds and point coordinates, copy them to GPU for hull checking, then copy results back for final subset enumeration on CPU. This serialization meant that while the GPU was computing one batch, the CPU sat idle waiting for results (and vice versa). We never achieved true concurrency.

Adding GPU hardware thus increased total resource capacity, but our pipeline never overlapped CPU and GPU work, resulting in end-to-end throughput only marginally better (or worse) than the 3-thread CPU version.

4. Proposed Solutions and Future Directions

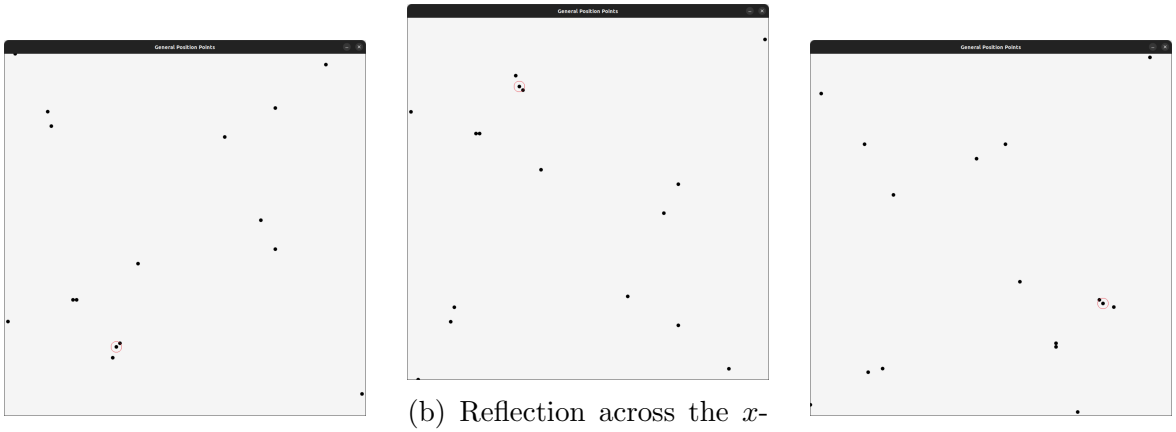
1. **Decouple stages via concurrent processes and file-based IPC:**
   - *Generator process*: continuously writes raw point-sets into a rotating log or memory-mapped file.
   - *GPU process*: tails the file, batches samples for hull checks on GPU (including the $|\text{hull}| \geq 6$ test), and writes boolean flags back.
   - *Verifier process*: reads the "false" flags (no convex-hexagon) and performs the slower $\binom{|\text{hull}|}{6}$ subset test on CPU.
2. **Ground-up GPU insertion algorithm:** Implement an insertion-based convex-hull algorithm (e.g., QuickHull) optimized for data parallelism, structuring work as independent "farthest-point" kernels to saturate SMs.
3. **Kernel fusion and persistent threads:** Launch a fixed pool of persistent threads that loop over many sample sets, pulling work from a global queue to avoid repeated kernel launches; overlap data transfer and compute via CUDA streams and pinned host memory.

5. Dataset Augmentation via Geometric Symmetries

We exploited the dihedral group D_4 (the eight isometries of the square) to enlarge our dataset while preserving colinearity, orientation, and convexity. Starting from 200 base sets of 16 points without any convex-hexagon, we applied the six nontrivial transforms:

1. Reflection across the x-axis
2. Reflection across the y-axis
3. Reflection across the line $y = x$
4. Rotation by 90°
5. Rotation by 180°
6. Rotation by 270°

This yields $200 \times 7 = 1400$ sets (including the identity transform).



(a) Original 16-point set

(b) Reflection across the x-axis

(c) Rotation by 90°

Figure 1: Examples of geometric augmentations. Note: the red circle is meant to highlight the same point in each set for ease of readability

6. Next Steps in Data Augmentation

- **Non-uniform scaling:** verify that affine maps (with $\alpha_x \neq \alpha_y$) preserve the negative label, since affine transforms preserve convex combinations.
- **Graph-based perturbations:** treat each set as a complete geometric graph, perturb edge weights by ϵ-bounded noise, and train for robustness.

7. Comparative Analysis of Transformer Architectures

This section evaluates several transformer-based models for extending point-generation from 25 to 32 points under non-colinearity and convexity constraints, given a masked 100×100 image scaled to 1000×1000.

7.1 Masked Autoencoder (MAE) Architecture

MAEs reconstruct masked image regions via an encoder-decoder transformer. The input is divided into patches, a high ratio of which are masked, then only visible patches pass through the encoder. The decoder reconstructs missing patches.

Key Components:

- Non-overlapping image patches with patch embedding and positional encodings
- Transformer-based encoder-decoder structure

Strengths:

- Robust self-supervised learning
- Scalable to large unlabeled datasets

Weaknesses:

- Sensitive to patch size and masking ratio
- Limited direct constraint integration for geometric tasks

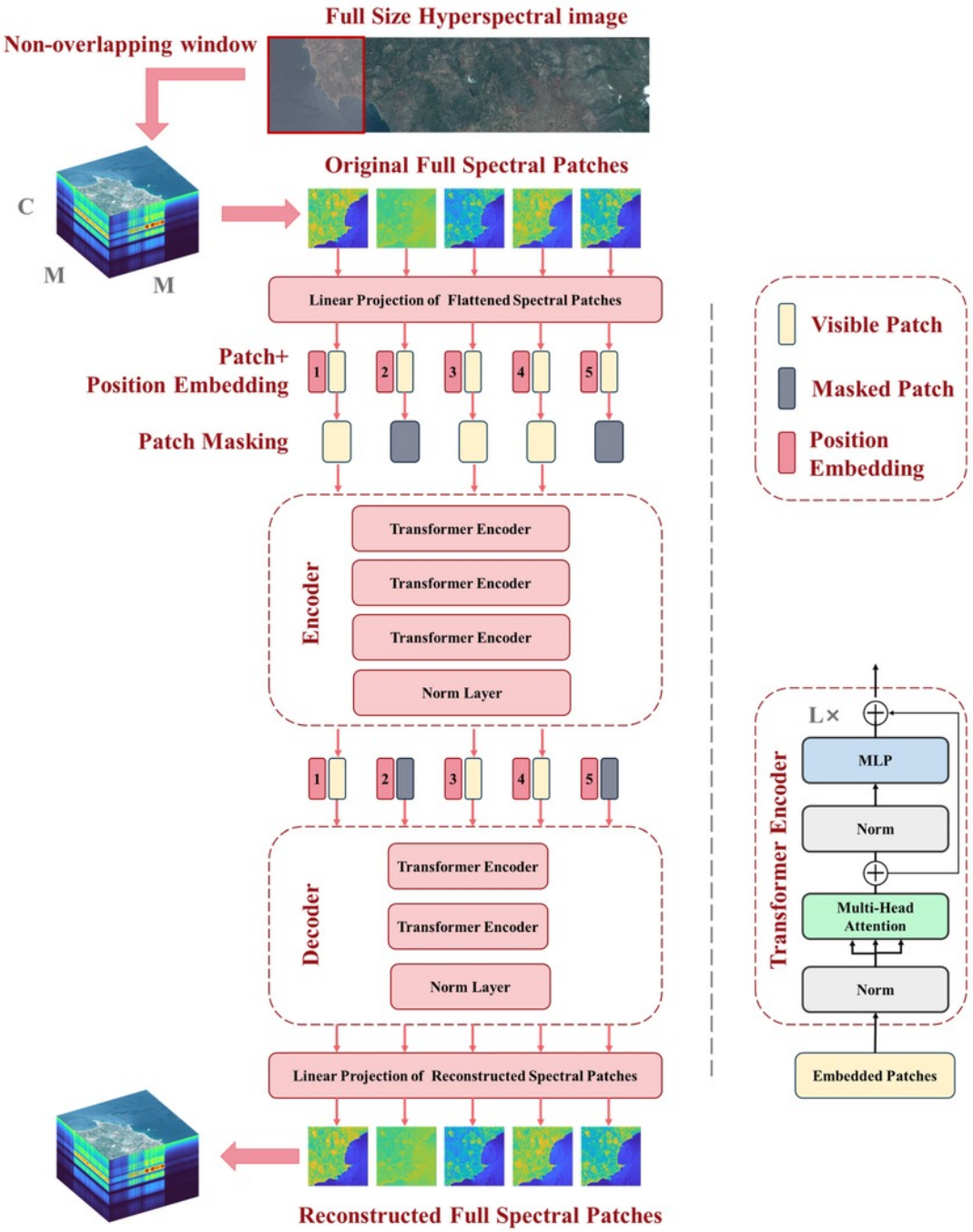


Figure 2: Masked Autoencoder Architecture

7.2 Masked Image Input Architecture

Here, masked regions are part of the input, and the model (often ViT-based) completes them. It excels at tasks treating structured point sets as partial visual data.

Benefits:

- Strong spatial reasoning via attention
- Predicts plausible geometric completions

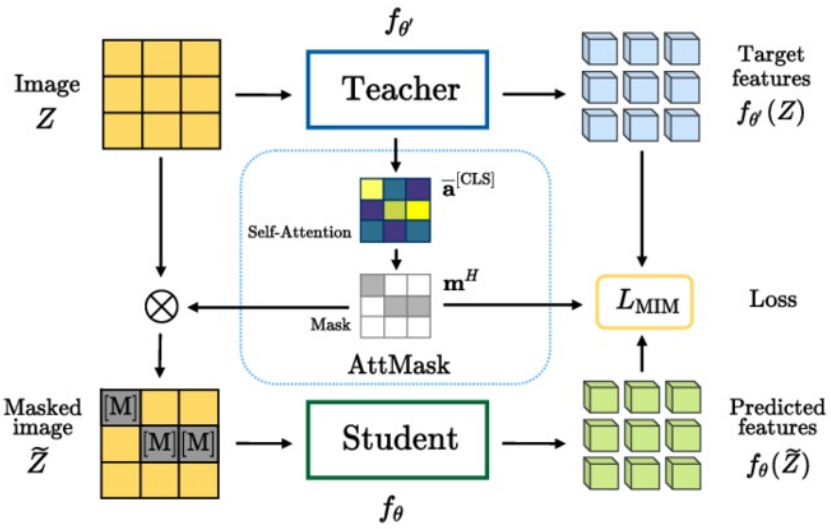


Figure 3: Masked Image Input Model (ViT-based)

7.3 Vision Transformers (ViT)

ViTs embed image patches and process them in a transformer encoder, enabling global patch interactions from early layers—critical for inferring missing points based on holistic context.

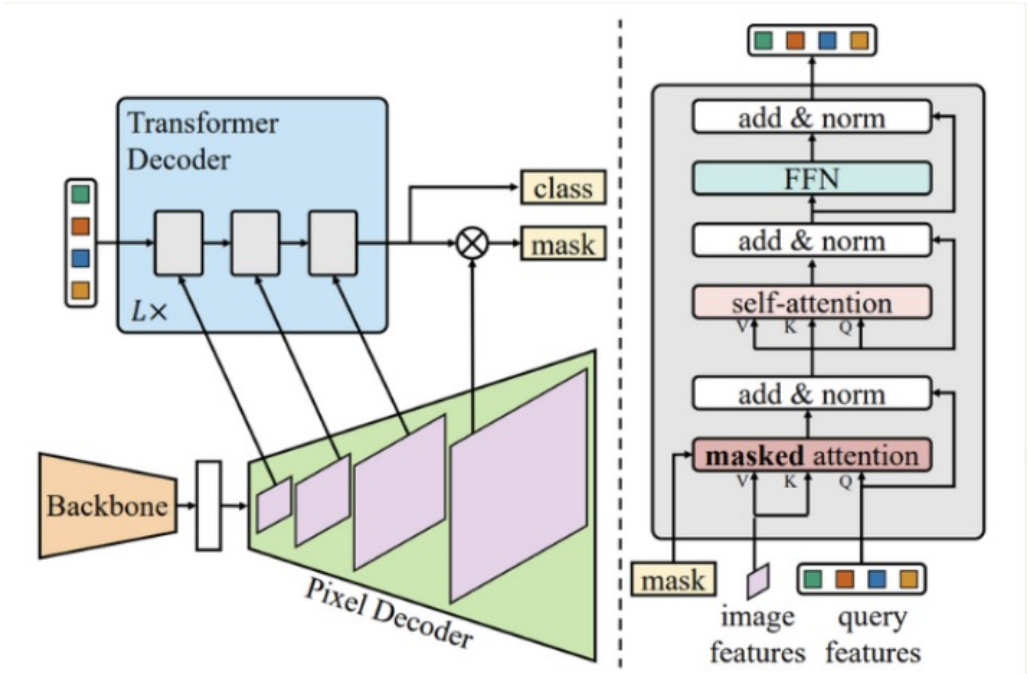


Figure 4: Vision Transformer Architecture

7.4 Conditional Diffusion Models

These learn data generation by reversing a noise process; conditional variants enforce constraints (e.g., convexity) during denoising iterations.

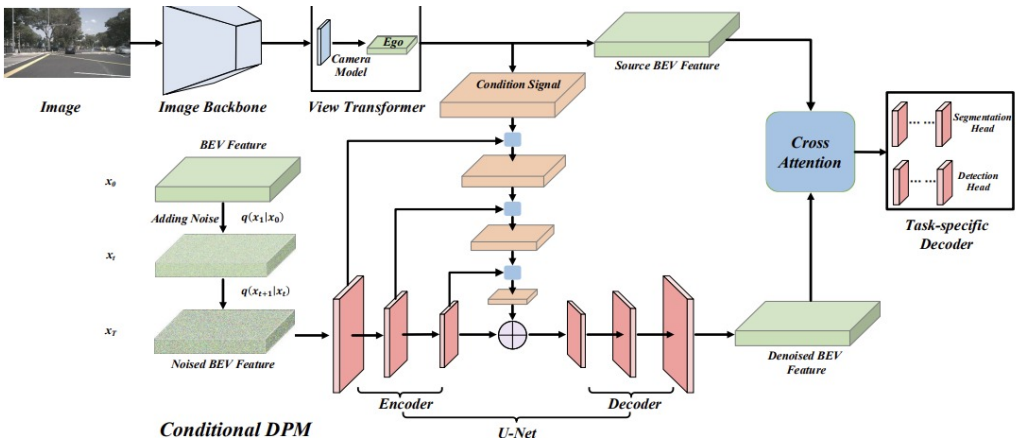


Figure 5: Conditional Diffusion Model

8. Incremental Construction Strategy

Our pipeline uses a 25-point seed image, which the model extends to 32 points satisfying:

- No three points co-linear
- Entire set forming a convex polygon

Images are scaled to 1000×1000, and CUDA is leveraged for acceleration. Post-generation, we apply geometric validation and augmentation.

9. Comparison of Architectures

Feature	**MAE**	**Masked Image Input**	**Conditional Diffusion**
Data Requirement	Moderate	Moderate	High
Contextual Learning	Local	Global	Global + Conditional
Point Generation Suitability	Low	High	High
Constraint Integration	Weak	Moderate	Strong
Computation Time	Low	Moderate	High
Interpretability	Moderate	High	Moderate

10. Conclusion

We will continue to improve the efficiency of our generation code and will aim for double our current dataset size as a start for transformer training which we will start the upcoming week. Among evaluated models, the Masked Image Input architecture (ViT-based) strikes the best balance between performance, constraint enforcement, and data efficiency. Future work includes dataset expansion, fine-tuning diffusion variants, and integrating learned geometric priors into the insertion pipeline.