MidcurveLLM: Geometric Dimension Reduction using Large language models

Yogesh H. Kulkarni, yogeshkulkarni@yahoo.com, http://orcid.org/0000-0001-5431-5727

Abstract

Lower-dimensional representations of shapes are essential for numerous applications. The midcurve, a one-dimensional (1D) representation of a two-dimensional (2D) planar shape, serves as a concise and informative descriptor. Widely utilized in animation, shape matching, retrieval, finite element analysis, and other fields, midcurves offer a compact means of capturing geometric information. This paper explores diverse methods for midcurve computation, considering various input shape types (e.g., images, sketches) and processing techniques (thinning, Medial Axis Transform (MAT), Chordal Axis Transform (CAT), Straight Skeletons).

This paper further investigates the integration of Large Language Models (LLMs) for computing midcurves in 2D geometric profiles. LLMs offer a promising avenue for distilling complex shapes while retaining key geometric details. Notably, traditional methods like Medial Axis Transform and Chordal Axis Transform often encounter difficulties with intricately shaped profiles and diverse connections, prompting a thorough evaluation of LLMs' potential efficacy in this domain.

This research tackles the challenge of midcurve extraction from 2D geometric profiles, framing it as a Graph Summarization problem analogous to text summarization. It meticulously addresses the specific challenges inherent in this domain, including geometric representation, variable-length input profiles, and branched midcurves. It explores neural network-based approaches such as Sequence-to-Sequence and Image-to-Image models, highlighting their strengths and limitations. Promisingly, Text-to-Text transformation using Large Language Models (LLMs) emerges as a novel and effective approach, demonstrated through prompt-based evaluations. This study contributes significantly to the advancement of midcurve computation, paving the way for efficient and accurate representation of complex 2D shapes.

Fine-tuning LLMs using a Boundary Representation (Brep) structure demonstrates potential for overcoming the limitations inherent in sequential point lists. However, challenges remain in LLM fine-tuning, ranging from model quality to dataset size, demanding further investigation and refinement. The complex nature of midcurve generation necessitates ongoing exploration, particularly in conjunction with the evolving capabilities of LLMs. This research opens exciting avenues for scholars and practitioners to delve deeper into the realm of geometric dimension reduction using LLMs, paving the way for significant advancements in shape analysis and representation.

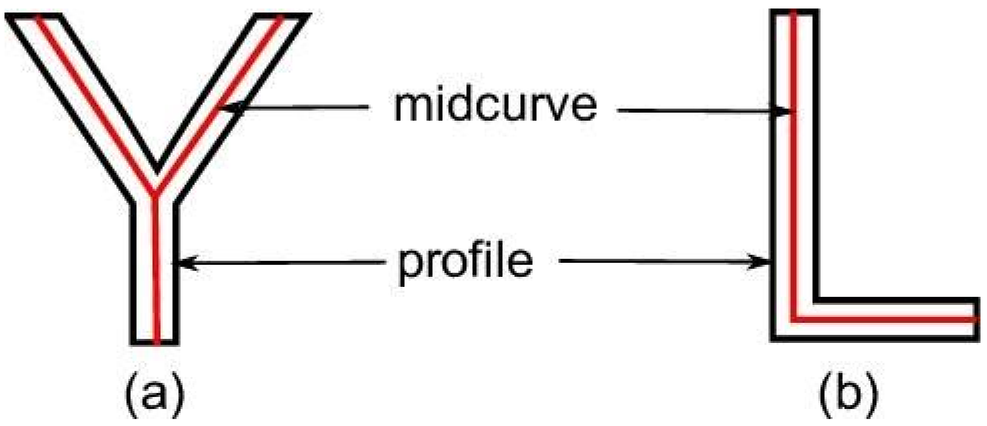
Introduction

In the field of Computer-Aided Design (CAD), particularly for thin-walled solids like sheet metal and plastic components, dimensional reduction plays a crucial role in expediting Computer-Aided Engineering (CAE) analysis. This is achieved through the creation of Mid-surfaces, defined as surfaces situated equidistantly between the bounding surfaces of the original solid. These Mid-surfaces serve as effective surrogates for analysis, significantly reducing computational complexity. Despite their importance, Mid-surface generation remains a laborious and predominantly manual process due to the scarcity of robust and efficient automated methodologies (Kulkarni 2021, Kulkarni 2022)

Existing automatic approaches for Mid-surface generation frequently encounter difficulties, leading to problematic artifacts such as gaps, missing patches, and overlapping surfaces. The manual correction of these errors can be incredibly time-consuming, often requiring hours or even days of tedious work. This highlights the urgent need for a reliable and efficient automated process for generating accurate and robust Mid-surfaces.

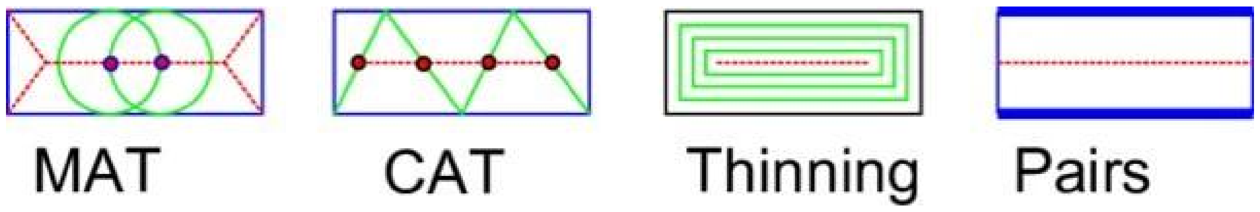
The concept of Mid-surfaces extends to two-dimensional (2D) planar shapes, where the equivalent is known as the Midcurve. Defined as a curve equidistant from the bounding curves of a 2D profile, the Midcurve provides a simplified representation that retains essential geometric information (ref Fig 1). While various traditional methods exist for Midcurve computation, the challenge of accurately representing complex shapes with diverse connections remains persistent.

The MidcurveNN research project investigates the potential of employing Neural Networks, specifically Large Language Models (LLMs), for tackling the challenging task of Midcurve generation. LLMs offer a powerful and versatile tool for processing complex information, and their potential to revolutionize the field of Midcurve computation is being thoroughly explored within this research framework. We aim to address the limitations of existing methods and establish a robust and efficient workflow for generating accurate and reliable Midcurves for diverse 2D geometric profiles.



(Fig. 1: Example of Midcurves of 2D Geometric profiles [ref])

Despite decades of research and the development of numerous approaches (ref Fig 2) like Medial Axis Transform (MAT), Chordal Axis Transform (CAT), Thinning, Pairing, and others, the problem of automatically generating accurate and robust midcurves for 2D geometric profiles remains unsolved. This persistent challenge stems from the inherent complexity of shapes and the diverse types of connections they can exhibit.



(Fig 2. Midcurve Approaches [ref])

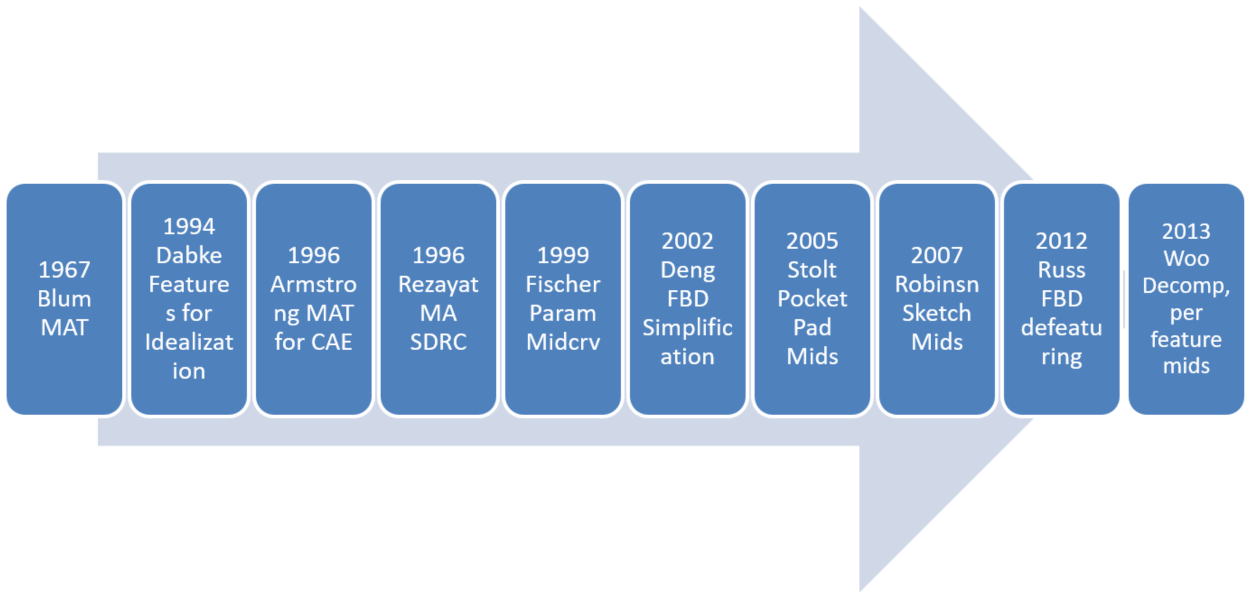
The MidcurveNN research project (Github repository (Kulkarni)) seeks to address this longstanding problem by exploring the potential of Neural Networks (NNs) in midcurve generation. Specifically, we investigate the capabilities of Large Language Models (LLMs), a powerful class of NNs, to learn and reproduce the intricate relationships between geometric features and their corresponding midcurves.

Traditional methods often encounter limitations when faced with complex shapes and diverse connections. LLMs, on the other hand, offer a unique opportunity to overcome these limitations due to their ability to process and analyze large amounts of data, including complex geometric representations. By leveraging the power of LLMs, MidcurveNN aims to establish a robust and efficient framework for automatic midcurve generation, paving the way for significant advancements in shape analysis and representation.

.

Related Work

The quest for efficient and robust methods for computing Mid-surfaces and Midcurves has spanned several decades, with significant advancements documented in Figure 1. A comprehensive review of the field is available in the survey paper (Kulkarni and Deshpande 2010).


1967 Blum MAT
1994 Dabke Features for Idealization
1996 Armstrong MAT for CAE
1996 Rezayat MA SDRC
1999 Fischer Param Midcrv
2002 Deng FBD Simplification
2005 Stolt Pocket Pad Mids
2007 Robinsn Sketch Mids
2012 Russ FBD defeaturing
2013 Woo Decomp, per feature mids

(Fig 3. Midcurve Research Timeline [ref])

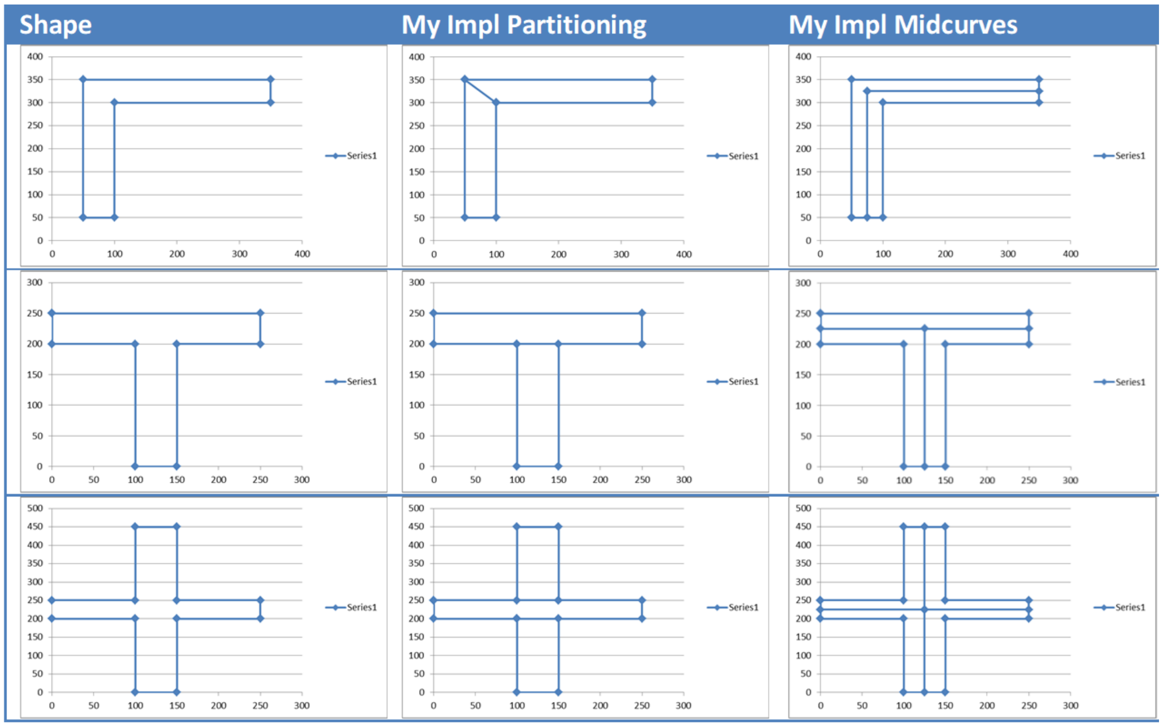
Classical Approaches

Several key observations emerge from the analysis of classical approaches:

1. Formal methods: Approaches like Medial Axis Transform (MAT) offer the advantage of being applicable to shapes of varying thickness and provide a reversible process for reconstructing the original shape. However, MAT and Thinning methods suffer from limitations such as generating unnecessary branches, reducing shape size, and being sensitive to changes in base geometry. Additionally, MAT-based approaches can be computationally expensive and require recomputation with altered base geometries, leading to potentially divergent results.
2. Heuristic methods: Chordal Axis Transform (CAT) approaches require pre-generated meshes, which can be challenging for complex 2D profiles. Thinning approaches, relying on straight line skeleton split events, can yield counterintuitive outcomes with sharp reflex vertices. The Pairs approach faces

difficulties in maintaining continuity when the input curves or surfaces are not in one-to-one form. Moreover, the quality of the generated Midcurve depends heavily on the sampled input shape points. Finally, Midcurve by profile decomposition, while not widely adopted, suffers from issues of generating redundant sub-shapes, rendering it ineffective and unstable for further processing.

Prior research has generally discouraged the use of formal methods like MAT and Thinning for Midcurve computation due to their reliance on heavy post-processing to eliminate unwanted curves. Similarly, heuristic methods, particularly decomposition, have been error-prone and inefficient. While the author's prior work (Figure 4) demonstrates success on simpler shapes, it suffers from scalability limitations due to its rule-based nature.



(Fig 4. Midcurve using Cellular Decomposition [ref])

Recent Advances and Research Gaps

Recent advancements have seen the integration of Neural Network-based techniques for Skeleton Midcurve computation, primarily through image processing. Notable examples include:

- Rodas, Neumann, and Wimmer 2020: Employed Deep Convolutional Neural Networks to simultaneously compute object boundaries and skeletons.
- Shen, Zhou, and Zeng 2021: Developed a Convolutional neural network-based skeleton extractor capturing both local and non-local image context.
- Shen, Zhou, and Zeng 2021: Introduced a hierarchical feature learning mechanism for skeleton extraction.
- Wang et al. 2023: Focused on content flux in skeletons as a learning objective.
- Panichev: Utilized U-net to extract skeletons, although not for Midcurve generation.

Despite these advancements, a review of the state-of-the-art emphasizes the need for Midcurve computation approaches that consider several key factors:

- Application context: The choice of method should be guided by the specific application and the desired level of accuracy.
- Shape characteristics: Different approaches may be better suited for specific types of shapes, such as primitive-shaped, thin sub-polygons.
- Aspect ratio: The aspect ratio of sub-polygons can significantly impact the effectiveness of various methods.

The current research gap lies in the need for efficient and accurate Midcurve generation for primitive-shaped, thin sub-polygons. Existing methods often struggle with non-primitive, skewed shapes and may yield inappropriate Midcurves.

Proposed Method

This paper proposes a novel approach to Midcurve generation using Deep Learning, specifically Large Language Models (LLMs). LLMs offer a unique opportunity to leverage their powerful language processing capabilities to learn and represent complex geometric relationships between 2D profiles and their corresponding Midcurves. This approach promises to overcome the limitations of existing methods and offer a robust and efficient solution for Midcurve computation

Problem Statement

Given a 2D closed shape (simple, convex, closed polygon) represented as a set of points or connected lines, find its corresponding midcurve (polyline, either closed or open).

The midcurve should be generated while preserving the key geometric properties of the original shape, similar to how text summarization retains the essence of a longer text document. This translates to the problem of Graph Summarization/Dimension-Reduction/Compression, where we aim to reduce the complexity of a larger graph (representing the polygon) to a smaller, simplified graph (representing the midcurve) while maintaining its underlying structure.

Geometry Representation

A fundamental challenge in employing Neural Networks for midcurve generation lies in the inherent differences between geometric shapes and the vector representations required by these models. Traditional encoders and decoders rely on fixed-size input and output vectors, posing difficulties for variable-length geometric shapes like polygons and polylines.

Several key issues prevent modeling shapes as simple sequences:

- Non-sequential nature: While a polygon may appear as a sequence of points, it lacks the inherent order and directionality of true sequential data.
- Incomplete representation: Shapes like Y or concentric circles cannot be drawn without lifting a pencil, violating the fundamental principle of sequence generation.
- Spatial information loss: Converting geometric shapes to vectors leads to information loss, particularly regarding spatial relationships and relative positions.

Existing graph-based representations, while addressing connectivity, fall short in capturing the crucial spatial information necessary for accurate midcurve generation. Nodes in such graphs carry coordinates, but the arcs lack inherent geometry (e.g., curvature, line segments). This limits the effectiveness of Graph Neural Networks, which rely on convolution and pooling operations that are not well-suited for handling both topology and geometry.

Therefore, the research requires significant innovation in geometric-graph embedding techniques. Such methods should enable convolution operations at nodes while incorporating the geometry of arcs (e.g., sets of point coordinates). Additionally, a novel pooling strategy needs to be developed to effectively aggregate information from neighboring nodes, arcs, and their respective geometric features into a single, meaningful representation. Addressing these challenges is crucial for enabling Neural Networks to learn and process the complex geometric relationships underlying 2D shapes and their corresponding midcurves.

Variable Lengths Issue

Midcurve generation presents a unique challenge in the domain of Neural Network-based dimension reduction. Unlike traditional sequence-to-sequence tasks, this problem involves variable-length inputs and outputs. The input sketch profile, which is a 2D parametric representation, can have varying numbers of points and line segments. Similarly, the output midcurve, being a 1D representation, exhibits variable length and may even have branches, deviating from the linear structure expected by standard encoders and decoders.

This variable length characteristic poses significant difficulties for existing deep learning libraries like TensorFlow, which require fixed-size input and output vectors. Padding approaches, commonly employed for handling variable lengths, prove problematic in this scenario. Traditional padding values like "0, 0" would be misconstrued as valid points within the geometric representation of the shape.

Furthermore, the non-linear connections present in both input polygons and branched midcurves render them unsuitable for typical Sequence-to-Sequence (seq2seq) networks. These networks rely on the linear order of data, which is not present in our domain due to potential loops and branches.

To address these challenges, alternative approaches need to be explored. One potential solution involves representing both the input and output using a fixed-size image format like 64x64 pixels. By coloring the profile and midcurve in the corresponding image bitmap, we can leverage the established capabilities of LSTM encoders and decoders for seq2seq processing. This approach allows for data augmentation through shifting, rotating, and scaling both input and output images, further enriching the training data.

However, this approach is limited to 2D sketch profiles with only linear segments and single, simple polygons without holes. The question of vectorization remains crucial. Representing each point as a 2D vector leads to variable-length input and output vectors, posing challenges for training. While potential solutions like padding with "0, 0" or "NULL" values exist, their validity within the context of geometric representations needs further investigation.

Additionally, exploring different network architectures beyond simple feedforward networks is vital. Recurrent Neural Networks (RNNs) and specifically LSTMs are well-suited for handling sequential data and could offer improved performance. Exploring dedicated architectures designed for graph-based data with variable lengths and non-linear connections could also be a promising avenue.

Addressing the variable length issue is a crucial step in enabling Neural Networks to effectively learn and generate midcurves from diverse 2D shapes. By investigating alternative representations, data augmentation techniques, and specialized network architectures, we can pave the way for robust and accurate midcurve generation using deep learning.

Implementation

While representing geometric shapes as images addresses both the representation and variable-size issues, it introduces a significant dilution in information accuracy. True geometric shapes are vector-based, whereas images are raster-based. This introduces an inherent approximation, which can potentially affect the quality of the generated midcurve. Additionally, the predicted output from an image-based model likely requires post-processing to convert it back into a geometric form, introducing another layer of complexity and potential errors.

Therefore, this project proposes a three-phase approach to address these challenges:

Phase I: Image to Image Transformation Learning:

- Img2Img: This phase focuses on learning image-to-image transformation using fixed-size 100x100 bitmaps.
- Data Augmentation: The training data will be augmented by scaling, rotating, and translating both input and output shapes within the fixed size.
- Network Architecture: An Encoder-Decoder network, specifically Semantic Segmentation or Pix2Pix, will be employed for image-based dimension reduction.

Phase II: Text to Text Transformation Learning:

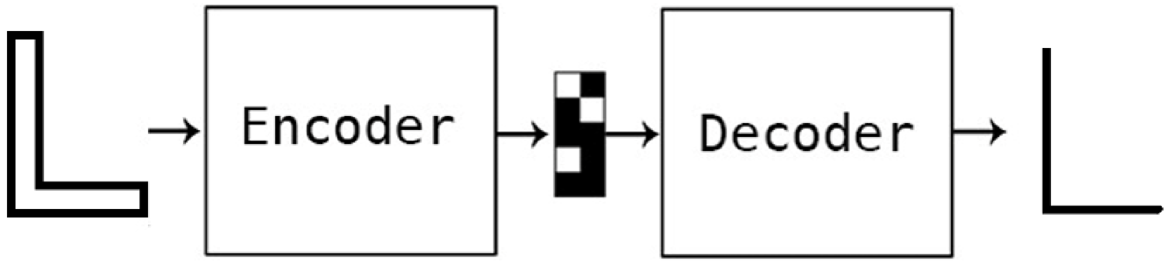
- Geometric Representation: A text-based representation of the geometry/graph/network will be explored to leverage Natural Language Processing (NLP) techniques.
- Existing Methods: The paper (Fatemi, Halcrow, and Bryan 2023) surveys several text-based representations for graph data, but none appear specifically suited for geometric shapes.
- Proposed Approach: A geometry representation similar to 2D Boundary Representation (B-rep) will be utilized, adapting the concept from 3D to 2D.

Phase III: Geometry to Geometry Transformation Learning:

- Graph Representation: Both input and output polyline graphs will be constructed with (x,y) coordinates as node features and edge information represented by node ID pairs.
- Polyline vs. Curve Representation: For polylines, edges are straight lines, eliminating the need for storing geometric intermediate points as features. For curves, a fixed number "n" of sampled points will be stored as features.
- Network Architecture: An Image-Segmentation-like Encoder-Decoder network will be implemented, incorporating Graph Convolution Layers from DGL in place of the usual 2D convolution layers typically used in image-based models.
- Data Generation: A variety of input-output polyline pairs will be generated using geometric transformations, avoiding the image-based transformations employed in Phase I.
- Potential Techniques: The use of Variational Graph Auto-Encoders (Cai) will be investigated for its potential benefits in this phase.

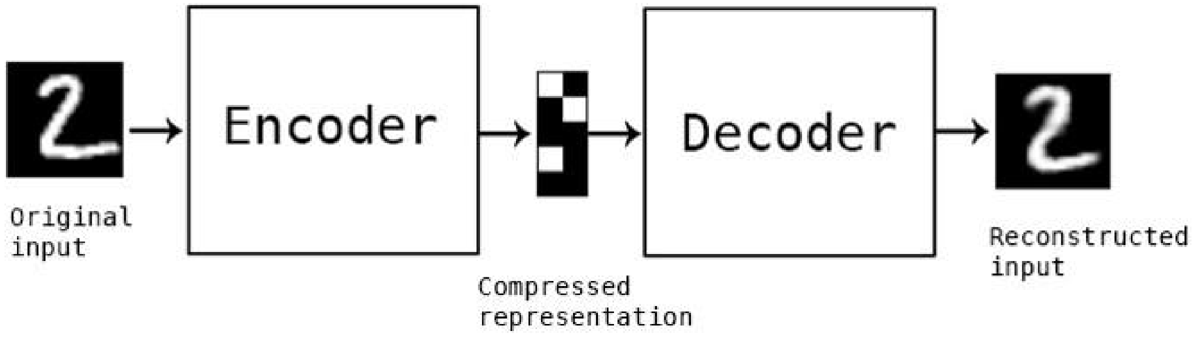
This phased approach aims to address the challenges of variable-size inputs and outputs while achieving accurate and robust midcurve generation by leveraging appropriate representation techniques and network architectures. Phase I lays the groundwork with a well-established approach in image-based processing, while Phase II explores the potential of text-based representations for geometric information. Phase III ultimately focuses on developing a network capable of directly processing and transforming geometric data, offering the most promising avenue for achieving high-fidelity midcurve generation.

Image-to-Image based Approach



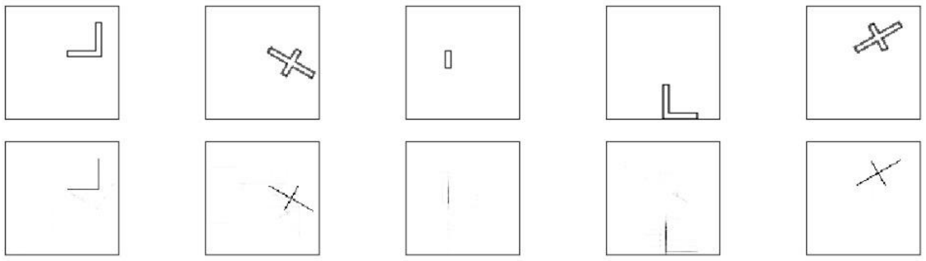
(Midcurve by Encoder Decoder [ref])

Such Transformers taking variable length input and output graphs have not been established. So, would taking this problem to image domain help?



(Auto Encoder on images [ref])

Various Image2Image networks have been tried by the MidcurveNN project but with limited success. Here is a sample:



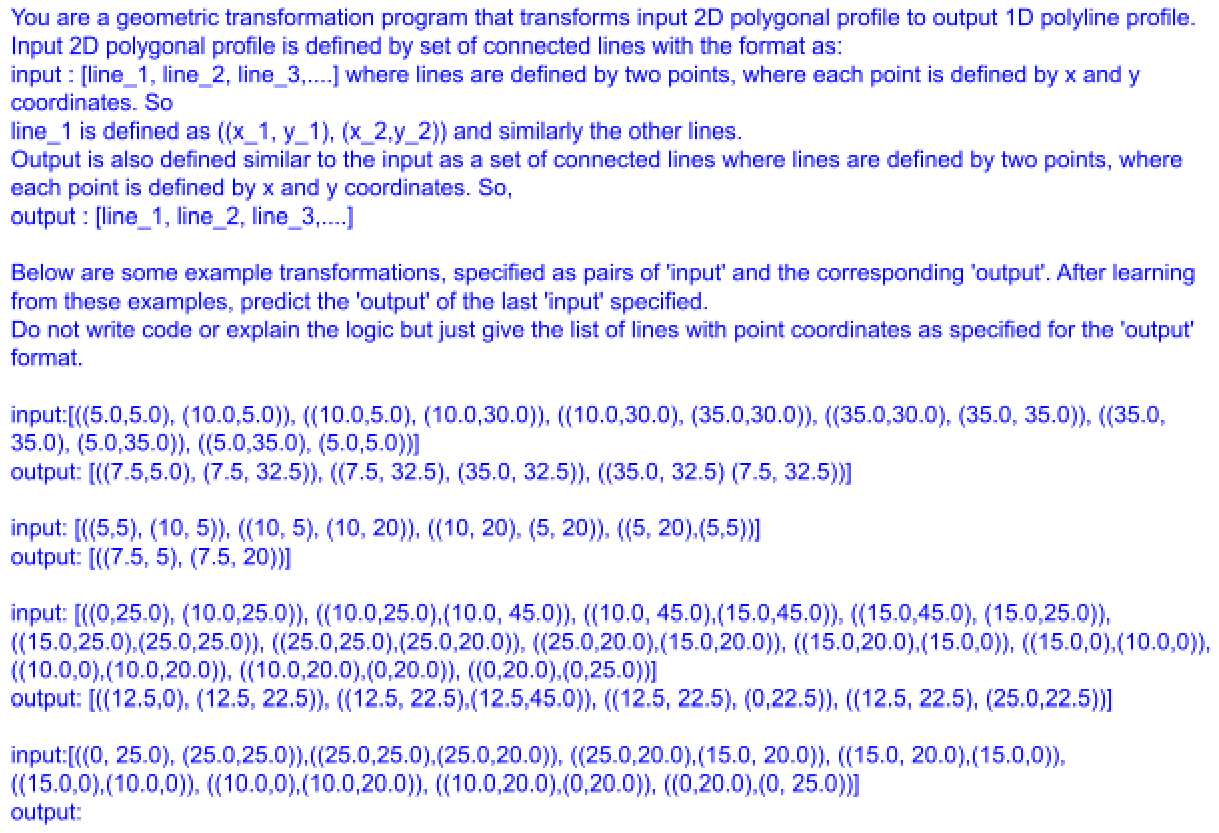
(Simple Encoder Decoder network in Tensorflow/Keras [ref])

Text-to-Text based Approach

Prompt-based Approach

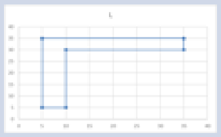
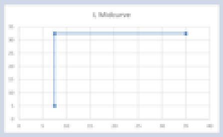
Evaluating LLMs (Large Language Models), like can be employed to generate the midcurve. Idea is to give a prompt telling what needs to be done along with some examples, and see if LLMs can generate shape for the test example. Geometry has been serialized into text as a simple list of points (More info (Kulkarni 2023)).

A few shots prompt developed:

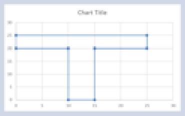
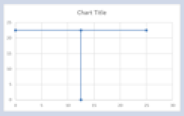


```
You are a geometric transformation program that transforms input 2D polygonal profile to output 1D polyline profile.
Input 2D polygonal profile is defined by set of connected lines with the format as:
input : [line_1, line_2, line_3,....] where lines are defined by two points, where each point is defined by x and y
coordinates. So
line_1 is defined as ((x_1, y_1), (x_2,y_2)) and similarly the other lines.
Output is also defined similar to the input as a set of connected lines where lines are defined by two points, where
each point is defined by x and y coordinates. So,
output : [line_1, line_2, line_3,....]

Below are some example transformations, specified as pairs of 'input' and the corresponding 'output'. After learning
from these examples, predict the 'output' of the last 'input' specified.
Do not write code or explain the logic but just give the list of lines with point coordinates as specified for the 'output'
format.

input:[((5.0,5.0), (10.0,5.0)), ((10.0,5.0), (10.0,30.0)), ((10.0,30.0), (35.0,30.0)), ((35.0,30.0), (35.0, 35.0)), ((35.0,
35.0), (5.0,35.0)), ((5.0,35.0), (5.0,5.0))]
output: [((7.5,5.0), (7.5, 32.5)), ((7.5, 32.5), (35.0, 32.5)), ((35.0, 32.5) (7.5, 32.5))]

input: [((5,5), (10, 5)), ((10, 5), (10, 20)), ((10, 20), (5, 20)), ((5, 20),(5,5))]
output: [((7.5, 5), (7.5, 20))]

input: [((0,25.0), (10.0,25.0)), ((10.0,25.0),(10.0, 45.0)), ((10.0, 45.0),(15.0,45.0)), ((15.0,45.0), (15.0,25.0)),
((15.0,25.0),(25.0,25.0)), ((25.0,25.0),(25.0,20.0)), ((25.0,20.0),(15.0,20.0)), ((15.0,20.0),(15.0,0)), ((15.0,0),(10.0,0)),
((10.0,0),(10.0,20.0)), ((10.0,20.0),(0,20.0)), ((0,20.0),(0,25.0))]
output: [((12.5,0), (12.5, 22.5)), ((12.5, 22.5),(12.5,45.0)), ((12.5, 22.5), (0,22.5)), ((12.5, 22.5), (25.0,22.5))]

input:[((0, 25.0), (25.0,25.0)),((25.0,25.0),(25.0,20.0)), ((25.0,20.0),(15.0, 20.0)), ((15.0, 20.0),(15.0,0)),
((15.0,0),(10.0,0)), ((10.0,0),(10.0,20.0)), ((10.0,20.0),(0,20.0)), ((0,20.0),(0, 25.0))]
output:
```

The first input example above represents an ‘L’ shape (shown below) and the second is an ‘I’, whereas the 3rd is a ‘Plus’ sign shape.

Profile Data		Profile Picture	Midcurve Data		Midcurve Picture
5.0	5.0		7.5	5.0	
10.0	5.0		7.5	32.5	
10.0	30.0		35.0	32.5	
35.0	30.0		7.5	32.5	
35.0	35.0				
5.0	35.0				

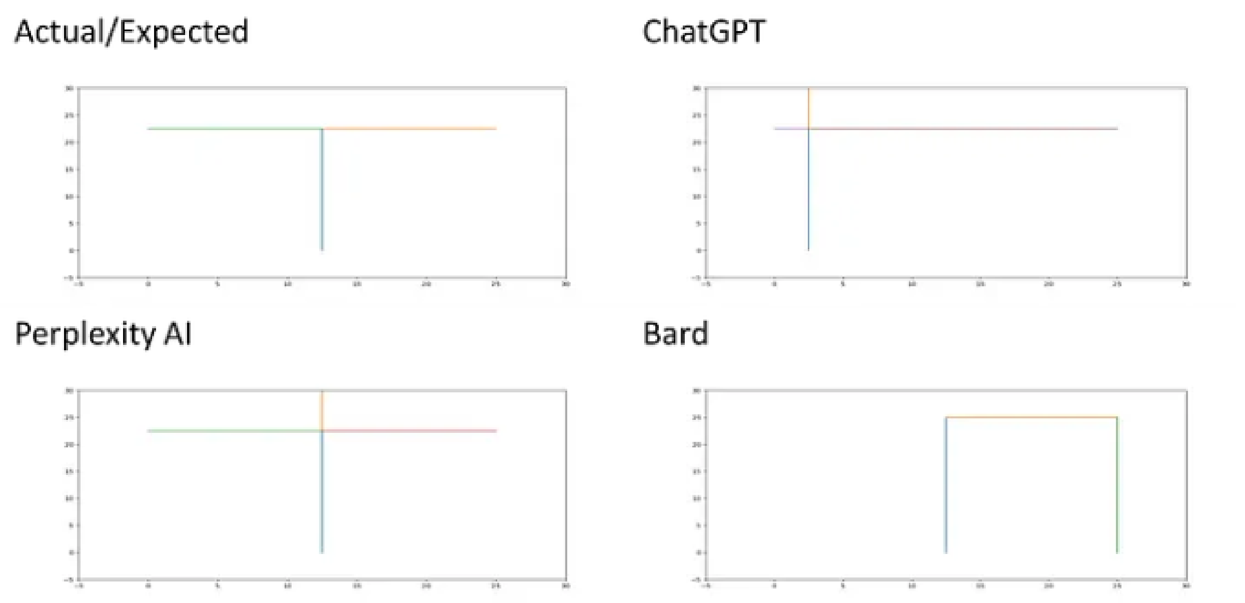
The last shape for which LLM has been asked for the answer is actually a ‘T’ shape. The picture below shows the correct/actual answer as well.

Profile Data		Profile Picture	Midcurve Data		Midcurve Picture
0	25.0		12.5	0	
25.0	25.0		12.5	22.5	
25.0	20.0		25.0	22.5	
15.0	20.0		0	22.5	
15.0	0				
10.0	0				
10.0	20.0				
0	20.0				

And the outputs computed by various LLMs (ChatGPT, Perplexity AI, Bard) , along with the real/actual answer:

```
Actual: [((12.5,0), (12.5,22.5)), ((12.5,22.5),(25.0,22.5)), ((12.5,22.5),(0,22.5))]
ChatGPT: [((2.5, 0), (2.5, 22.5)), ((2.5, 22.5), (2.5, 45.0)), ((2.5, 22.5), (25.0, 22.5)), ((2.5, 22.5), (12.5, 22.5)), ((2.5, 22.5), (0, 22.5)), ((2.5, 22.5), (25.0, 22.5))]
Perplexity: [((12.5,0), (12.5, 22.5)), ((12.5, 22.5),(12.5,45.0)), ((12.5, 22.5), (0,22.5)), ((12.5, 22.5), (25.0,22.5))]
Bard: [((12.5, 0), (12.5, 25.0)), ((12.5, 25.0), (25.0, 25.0)), ((25.0, 25.0), (25.0, 0))]
```

Visually here is how results from different LLMs look:



All of the above have failed. Even latest (Oct 2023), the results are:

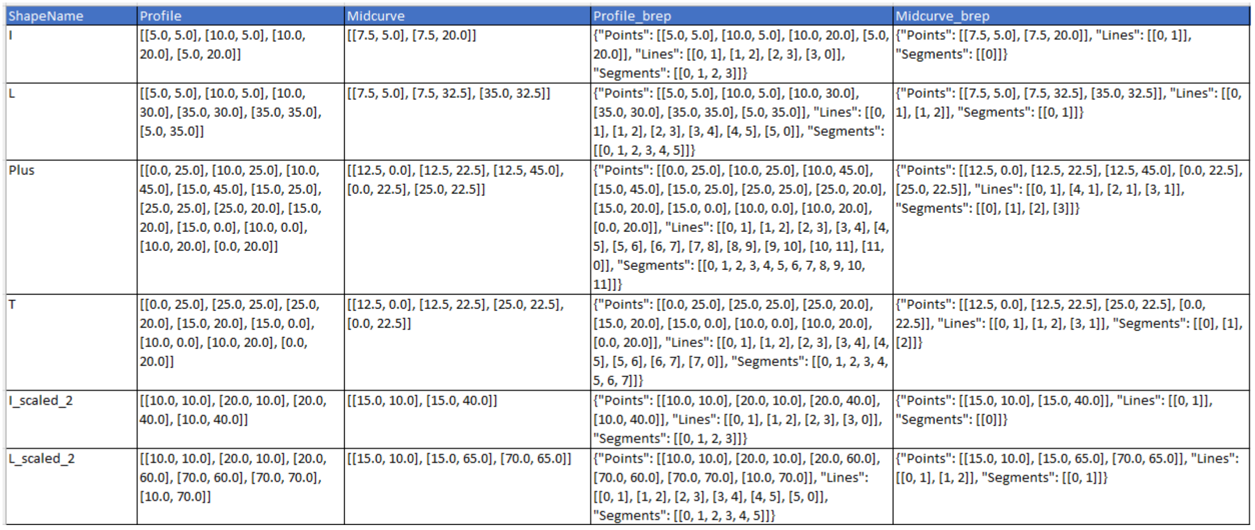
- llama 7B g4_0 ggml: (8, 17) & (64, 32): Wrong.
- Bard: [((8.33,5),(8.33, 22.5)), ((8.33, 22.5), (25,22.5)), ((8.33, 22.5), (0,25))]: Wrong.
- Hugging Chat: [((12.5, 0), (12.5, 22.5)), ((12.5, 22.5), (25.0, 22.5)), ((25.0, 22.5), (25.0, 25.0))]: a bit wrong on the last line.
- GPT-4: [((12.5,0), (12.5,22.5)), ((12.5,22.5),(0,22.5)), ((12.5,22.5),(25.0,22.5))] just change in sequence of lines, and that's inconsequential, so the answer is correct.
- Claude: [((12.5, 0.0), (12.5, 22.5)), ((12.5, 22.5), (12.5, 25.0)), ((12.5, 22.5), (0.0, 22.5)), ((12.5, 22.5), (25.0, 22.5))]

Although other proprietary and open-source models need to catch-up with GPT-4, even GPT-4 needs to be developed further to understand not just sequential lines but graphs/networks with different shapes, essentially, the geometry.

Fine-tuning based approach

One limitation of 2D geometry-shape, represented as a sequential list of points, is that we can not represent line intersections or concentric loops. To address this a more comprehensive structure has been proposed which is based on the corresponding 3D structure popular in Solid Modeling, called Brep (Boundary Representation).

```
{
          'ShapeName': 'I',
          'Profile': [(5.0, 5.0), (10.0, 5.0), (10.0, 20.0), (5.0, 20.0)],
          'Midcurve': [(7.5, 5.0), (7.5, 20.0)],
          'Profile_brep': {
                                    'Points': [(5.0, 5.0), (10.0, 5.0), (10.0, 20.0),(5.0, 20.0)], # list of (x,y) coordinates
                                    'Lines': [[0, 1], [1, 2], [2, 3], [3, 0]], # list of point ids (ie index in the Points list)
            'Segments': [[0, 1, 2, 3]] # list of line ids (ie index in Lines list)
                                    },
          'Midcurve_brep': {
                                    'Points': [(7.5, 5.0), (7.5, 20.0)],
                                    'Lines': [[0, 1]],
            'Segments': [[0]]
                                    },
}
```



ShapeName	Profile	Midcurve	Profile_brep	Midcurve_brep
I	[[5.0, 5.0], [10.0, 5.0], [10.0, 20.0], [5.0, 20.0]]	[[7.5, 5.0], [7.5, 20.0]]	{"Points": [[5.0, 5.0], [10.0, 5.0], [10.0, 20.0], [5.0, 20.0]], "Lines": [[0, 1], [1, 2], [2, 3], [3, 0]], "Segments": [[0, 1, 2, 3]]}	{"Points": [[7.5, 5.0], [7.5, 20.0]], "Lines": [[0, 1]], "Segments": [[0]]}
L	[[5.0, 5.0], [10.0, 5.0], [10.0, 30.0], [35.0, 30.0], [35.0, 35.0], [5.0, 35.0]]	[[7.5, 5.0], [7.5, 32.5], [35.0, 32.5]]	{"Points": [[5.0, 5.0], [10.0, 5.0], [10.0, 30.0], [35.0, 30.0], [35.0, 35.0], [5.0, 35.0]], "Lines": [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5], [5, 0]], "Segments": [[0, 1, 2, 3, 4, 5]]}	{"Points": [[7.5, 5.0], [7.5, 32.5], [35.0, 32.5]], "Lines": [[0, 1], [1, 2]], "Segments": [[0, 1]]}
Plus	[[0.0, 25.0], [10.0, 25.0], [10.0, 45.0], [15.0, 45.0], [15.0, 25.0], [25.0, 25.0], [25.0, 20.0], [15.0, 20.0], [15.0, 0.0], [10.0, 0.0], [10.0, 20.0], [0.0, 20.0]]	[[12.5, 0.0], [12.5, 22.5], [12.5, 45.0], [0.0, 22.5], [25.0, 22.5]]	{"Points": [[0.0, 25.0], [10.0, 25.0], [10.0, 45.0], [15.0, 45.0], [15.0, 25.0], [25.0, 25.0], [25.0, 20.0], [15.0, 20.0], [15.0, 0.0], [10.0, 0.0], [10.0, 20.0], [0.0, 20.0]], "Lines": [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7], [7, 8], [8, 9], [9, 10], [10, 11], [11, 0]], "Segments": [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]]}	{"Points": [[12.5, 0.0], [12.5, 22.5], [12.5, 45.0], [0.0, 22.5], [25.0, 22.5]], "Lines": [[0, 1], [4, 1], [2, 1], [3, 1]], "Segments": [[0], [1], [2], [3]]}
T	[[0.0, 25.0], [25.0, 25.0], [25.0, 20.0], [15.0, 20.0], [15.0, 0.0], [10.0, 0.0], [10.0, 20.0], [0.0, 20.0]]	[[12.5, 0.0], [12.5, 22.5], [25.0, 22.5], [0.0, 22.5]]	{"Points": [[0.0, 25.0], [25.0, 25.0], [25.0, 20.0], [15.0, 20.0], [15.0, 0.0], [10.0, 0.0], [10.0, 20.0], [0.0, 20.0]], "Lines": [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7], [7, 0]], "Segments": [[0, 1, 2, 3, 4, 5, 6, 7]]}	{"Points": [[12.5, 0.0], [12.5, 22.5], [25.0, 22.5], [0.0, 22.5]], "Lines": [[0, 1], [1, 2], [3, 1]], "Segments": [[0], [1], [2]]}
I_scaled_2	[[10.0, 10.0], [20.0, 10.0], [20.0, 40.0], [10.0, 40.0]]	[[15.0, 10.0], [15.0, 40.0]]	{"Points": [[10.0, 10.0], [20.0, 10.0], [20.0, 40.0], [10.0, 40.0]], "Lines": [[0, 1], [1, 2], [2, 3], [3, 0]], "Segments": [[0, 1, 2, 3]]}	{"Points": [[15.0, 10.0], [15.0, 40.0]], "Lines": [[0, 1]], "Segments": [[0]]}
L_scaled_2	[[10.0, 10.0], [20.0, 10.0], [20.0, 60.0], [70.0, 60.0], [70.0, 70.0], [10.0, 70.0]]	[[15.0, 10.0], [15.0, 65.0], [70.0, 65.0]]	{"Points": [[10.0, 10.0], [20.0, 10.0], [20.0, 60.0], [70.0, 60.0], [70.0, 70.0], [10.0, 70.0]], "Lines": [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5], [5, 0]], "Segments": [[0, 1, 2, 3, 4, 5]]}	{"Points": [[15.0, 10.0], [15.0, 65.0], [70.0, 65.0]], "Lines": [[0, 1], [1, 2]], "Segments": [[0, 1]]}

Column information is as below:

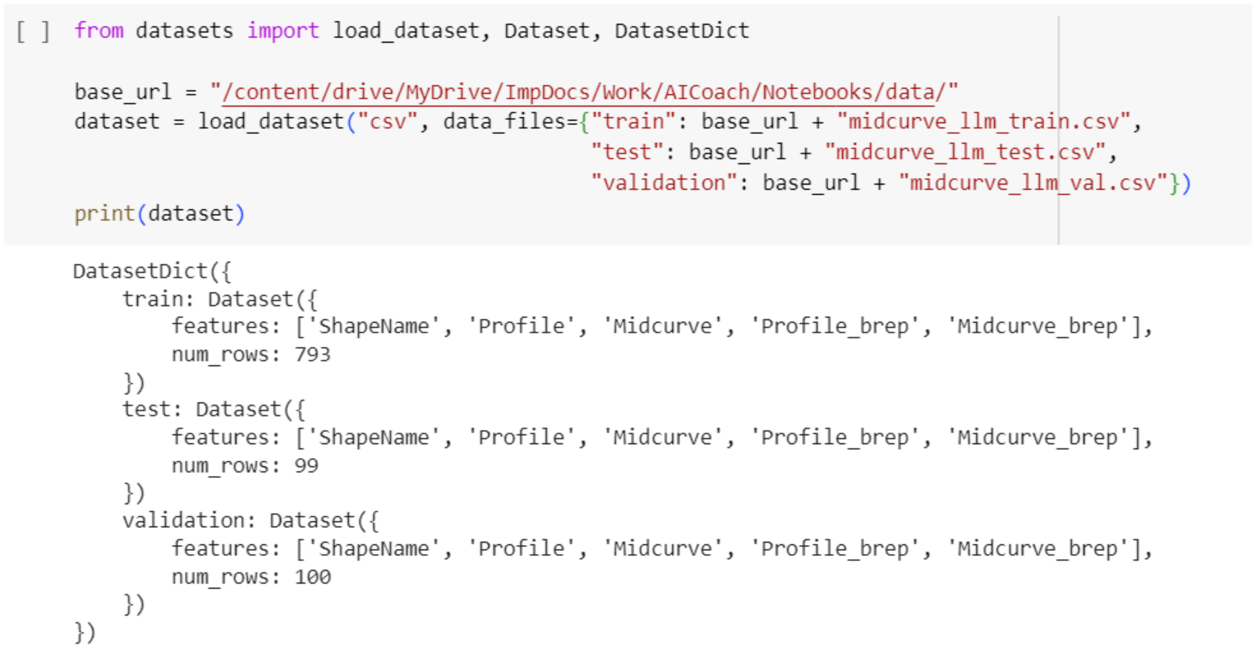
- ShapeName (text): name of the shape. Just for visualization/reports.
- Profile (text): List of Points coordinates (x,y) representing outer profile shape, typically closed.
- Midcurve (text): List of Points coordinates (x,y) representing inner midcurve shape, typically open.
- Profile_brep (text): Dictionary in Brep format representing outer profile shape, typically closed.
- Midcurve_brep (text): Dictionary in Brep format representing inner midcurve shape, typically open.

Each Segment is a continuous list of lines. In case of, say `Midcurve-T`, as there is an intersection, we can treat each line in a separate segment. In the case of 'Profile O', there will be two segments, one for outer lines and another for inner lines. Each line is a list of points, for now, linear. Points is a list of coordinates (x,y), later can be (x,y,z).

Once we have this Brep representations of both, profile and the corresponding midcurve, in the text form, then we can try various machine translation approaches or LLM based fine tuning or few-shots prompt engineering.

One major advantage of text based method over image based method is that image output still has stray pixels, cleaning which will be a complex task. But the text method has exact points. It may just give odd lines, which can be removed easily.

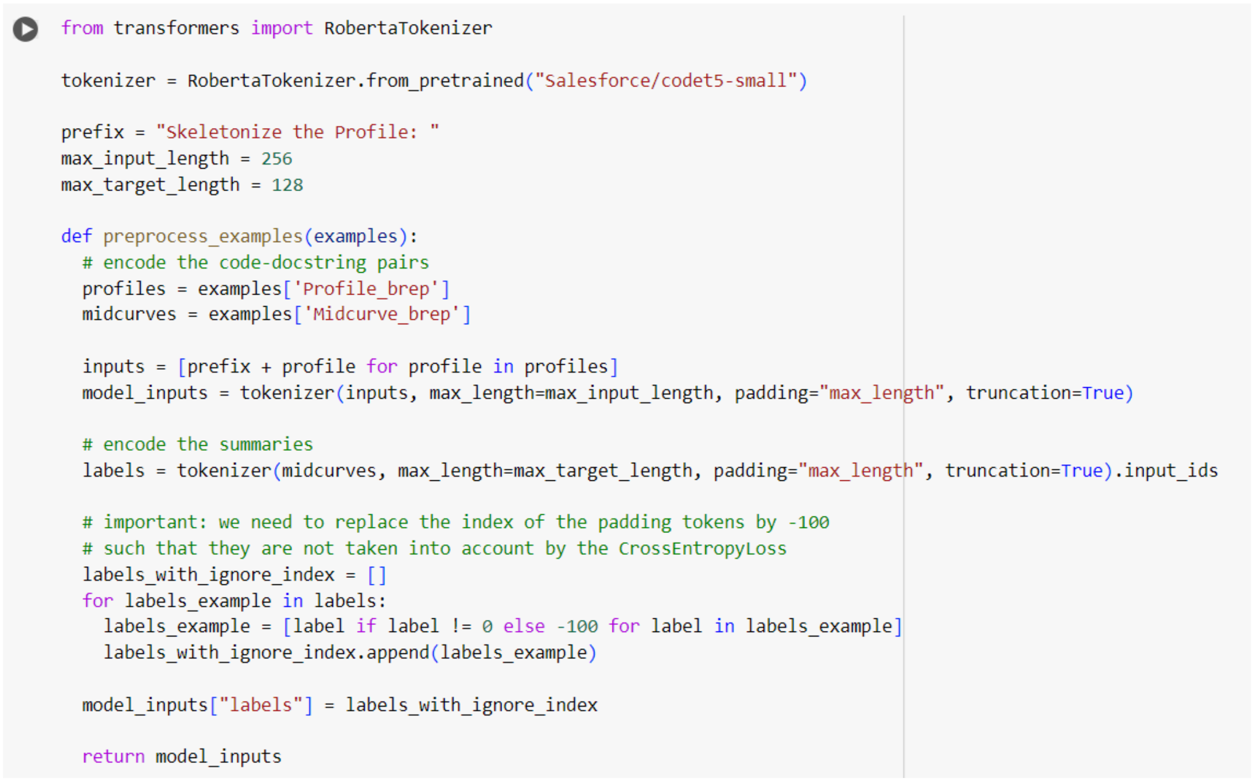
Dataset when imported looks like



```
[ ] from datasets import load_dataset, Dataset, DatasetDict

    base_url = "/content/drive/MyDrive/ImpDocs/Work/AICoach/Notebooks/data/"
    dataset = load_dataset("csv", data_files={"train": base_url + "midcurve_llm_train.csv",
                                              "test": base_url + "midcurve_llm_test.csv",
                                              "validation": base_url + "midcurve_llm_val.csv"})
    print(dataset)
```

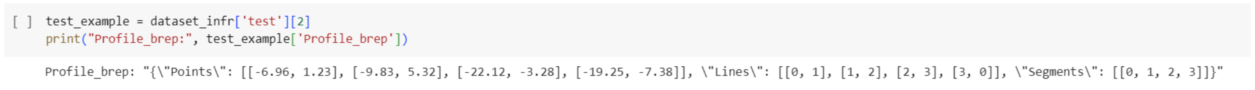
```
    DatasetDict({
        train: Dataset({
            features: ['ShapeName', 'Profile', 'Midcurve', 'Profile_brep', 'Midcurve_brep'],
            num_rows: 793
        })
        test: Dataset({
            features: ['ShapeName', 'Profile', 'Midcurve', 'Profile_brep', 'Midcurve_brep'],
            num_rows: 99
        })
        validation: Dataset({
            features: ['ShapeName', 'Profile', 'Midcurve', 'Profile_brep', 'Midcurve_brep'],
            num_rows: 100
        })
    })
```

Preprocessing is done to tokenize, generate ids and prepare attention.

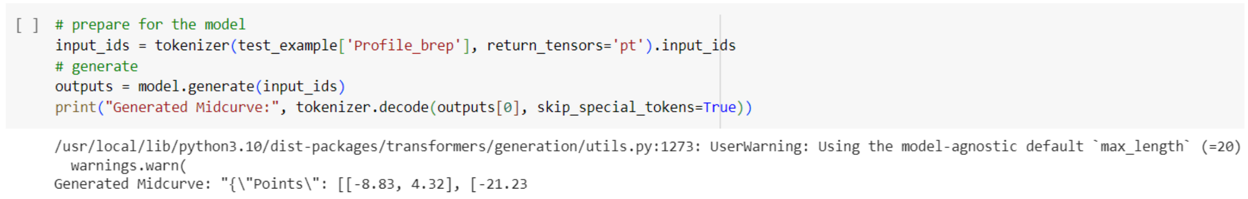


```python
from transformers import RobertaTokenizer

tokenizer = RobertaTokenizer.from_pretrained("Salesforce/codet5-small")

prefix = "Skeletonize the Profile: "
max_input_length = 256
max_target_length = 128

def preprocess_examples(examples):
  # encode the code-docstring pairs
  profiles = examples['Profile_brep']
  midcurves = examples['Midcurve_brep']

  inputs = [prefix + profile for profile in profiles]
  model_inputs = tokenizer(inputs, max_length=max_input_length, padding="max_length", truncation=True)

  # encode the summaries
  labels = tokenizer(midcurves, max_length=max_target_length, padding="max_length", truncation=True).input_ids

  # important: we need to replace the index of the padding tokens by -100
  # such that they are not taken into account by the CrossEntropyLoss
  labels_with_ignore_index = []
  for labels_example in labels:
    labels_example = [label if label != 0 else -100 for label in labels_example]
    labels_with_ignore_index.append(labels_example)

  model_inputs["labels"] = labels_with_ignore_index

  return model_inputs
```

The dataset is used to fine-tune the base model called `”Salesforce/codet5-small”.`
Trained model is saved and used for inference on test samples.

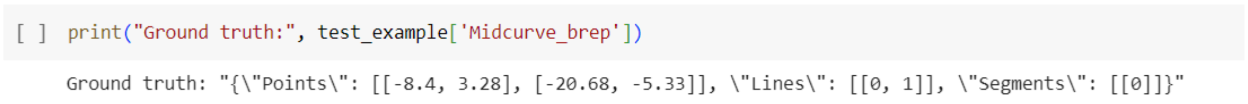


```
[ ] test_example = dataset_infr['test'][2]
    print("Profile_brep:", test_example['Profile_brep'])

    Profile_brep: "{\"Points\": [[-6.96, 1.23], [-9.83, 5.32], [-22.12, -3.28], [-19.25, -7.38]], \"Lines\": [[0, 1], [1, 2], [2, 3], [3, 0]], \"Segments\": [[0, 1, 2, 3]]}"
```



```
[ ] # prepare for the model
    input_ids = tokenizer(test_example['Profile_brep'], return_tensors='pt').input_ids
    # generate
    outputs = model.generate(input_ids)
    print("Generated Midcurve:", tokenizer.decode(outputs[0], skip_special_tokens=True))

    /usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:1273: UserWarning: Using the model-agnostic default `max_length` (=20)
      warnings.warn(
    Generated Midcurve: "{\"Points\": [[-8.83, 4.32], [-21.23
```

The output predicted far away from the corresponding ground truth.



```
[ ] print("Ground truth:", test_example['Midcurve_brep'])

    Ground truth: "{\"Points\": [[-8.4, 3.28], [-20.68, -5.33]], \"Lines\": [[0, 1]], \"Segments\": [[0]]}"
```

There could be many reasons, quality of LLM, training parameters, and above all, need for a far bigger dataset for fine-tuning.

Conclusions

This research not only contributes significantly to our understanding of midcurve computation but also illuminates the complex challenges and exciting potential avenues for future exploration. The study highlights the intricate interplay between established methodologies and cutting-edge approaches, particularly the integration of Large Language Models (LLMs) in midcurve generation. This underscores the nuanced nature of geometric dimension reduction and the need for innovative solutions to address the associated complexities.

The work clearly delineates the challenges inherent in variable-length input data, the representation of intricate shapes, and the limitations of existing models like Seq2Seq and Image2Image networks. These limitations necessitate a paradigm shift towards more advanced and adaptable methodologies.

The proposed MidcurveNN, utilizing an Encoder-Decoder architecture with supervised learning, demonstrates a promising direction for overcoming the hurdles plaguing traditional midcurve computation methods. The innovative application of Brep structures in fine-tuning LLMs represents a commendable effort towards achieving higher geometric precision in the generated midcurves.

However, the observed discrepancies between the predicted outputs and ground truths highlight the need for more extensive datasets and further refinement of the training parameters. This ongoing research underscores the crucial role of extensive training data and meticulous fine-tuning in achieving optimal performance with LLM-based approaches.

In essence, this research serves as a vital catalyst for the advancement of midcurve computation methodologies. The seamless integration of LLMs, the exploration of novel geometric representations, and the introspection into fine-tuning mechanisms collectively contribute to the ongoing dialogue in geometric dimension reduction. This work invites further scrutiny, refinement, and advancements, beckoning researchers and practitioners to delve deeper into the transformative intersection of geometry and advanced machine learning paradigms. As we navigate the intricate landscape of midcurve generation, the potential for groundbreaking discoveries and impactful applications remains vast and ever-evolving.

References

Cai, Z. (n.d.). *Variational Graph Auto-Encoders*. GitHub. Retrieved December 8, 2023, from https://github.com/dmlc/dgl/tree/master/examples/pytorch/vgae

Fatemi, B., Halcrow, J., & Bryan, P. (2023). Talk like a Graph: Encoding Graphs for Large Language Models. *arXiv*, 2310.04560. https://doi.org/10.48550/arXiv.2310.04560

Kulkarni, Y. H. (2021). MidcurveNN: Neural Network for Computing Midcurve of a Thin Polygon. *CAD Conference*, 223-225. 10.14733/cadconfP.2021.223-225

Kulkarni, Y. H. (2022). MidcurveNN: Neural Network for Computing Midcurve of a Thin Polygon. *Computer-aided Design & Applications*, *19*(6), 1154-1161. https://doi.org/10.14733/cadaps.2022.1154-1161

Kulkarni, Y. H. (2023). *MidcurveNN*. MidcurveNN. Retrieved Dec 8, 2023, from https://github.com/yogeshhk/MidcurveNN

Kulkarni, Y. H. (2023, May 31). Geometry, Graphs and GPT. *Medium*. https://medium.com/technology-hits/geometry-graphs-and-gpt-2862d6d24866

Kulkarni, Y. H. (2023, June 16). *MidcurveNN: Encoder-Decoder Neural Network for Computing Midcurve of a Thin Polygon*. ODSC India 2019. Retrieved December 8, 2023, from https://confengine.com/conferences/odsc-india-2019/proposal/10090/midcurvenn-encoder-decoder-neural-network-for-computing-midcurve-of-a-thin-polygon

Kulkarni, Y. H., & Deshpande, S. (2010). Medial Object Extraction - A State of the Art. *International Conference on Advances in Mechanical Engineering (ICAME), SVNIT*.

Panichev, I. (n.d.). Accurate and efficient skeleton extraction using a convolutional neural network. *arXiv preprint*. 2307.12410

Rodas, A., Neumann, L., & Wimmer, M. (2020, 4). Deep Learning for Skeleton Extraction from Foreground Objects in Images. *ACM Transactions on Graphics (TOG)*, *39*(4), 1 - 14. -

Shen, Z., Zhou, Z., & Zeng, Y. (2021, 1). A convolutional neural network-based skeleton extraction method for 3D shapes. *IEEE Transactions on Visualization and Computer Graphics*, *27*(11), 3729-3739. -

Wang, H., Zhou, B., Wang, C., & Liu, Y. (2023, - -). Learning content flux for skeletonization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, -(-), -. -