Information Technology Course
Module Software Engineering
by Damir Dobric / Andreas Pech

Implement the SDR representation in the MAUI application

Mehe Zabin Trisha
1400382
mehe.trisha@stud.fra-uas.de

Kazi Md. Abu Mithfa
1429088
kaziabu.mithfa@stud.fra-uas.de

Saleque Ahmed
l393172
saleque.ahmed@stud.fra-uas.de

Abstract— In the field of software engineering, there is a growing demand for user-friendly graphical interfaces, particularly in applications where visualizing data is crucial. This paper details the creation and deployment of a Multimodal Adaptive User Interface (MAUI) application designed to display Sparse Distributed Representations (SDRs) utilizing Scalable Vector Graphics (SVG). The project was motivated by the necessity to seamlessly transition SDR representations from supplied Python scripts to an interactive MAUI application using C#. Our strategy involved crafting an MAUI application capable of accepting input in various forms, including text and file formats, to enhance user convenience. Utilizing the features of SVG, our application generates SDR outputs in both horizontal and vertical layouts, accommodating diverse user preferences. Furthermore, we also introduce a download feature for the user of the application, by which the user can download the generated output image. Following extensive development and continuous work, our MAUI application faithfully replicates the functionalities of the expected outcome provided by our professor, successfully meeting the project's primary objective. Our efforts contribute to advancing SDR-based methodologies and the development of MAUI applications, paving the way for future research and innovation in software engineering.

Keywords—MAUI Application, SDR, Horizontal and Vertical Representations, SVG.

I. INTRODUCTION

Sparse Distributed Representation is a frequently employed idea in neural network frameworks, especially within unsupervised learning and spatial coding. It involves representing data in a high-dimensional space where only a small fraction of dimensions is given at any time. This representation is valuable for tasks such as pattern recognition, anomaly detection, and memory storage among others.

Ankur et al. [1] work on the unpredictable slowdown of worker nodes while executing large matrix computations such as matrix-vector multiplication in a distributed fashion. To solve the problem they use rate less code that generates redundant linear combinations of the matrix rows (or columns) and distribute them across workers. For allocating rows of a sparse matrix to workers they proposed a balance row allocation strategy.

Peixiang et al. [2] introduce a method to identify crucial points within power systems by utilizing measurable parameters from active distribution grids. By employing sensitive sparse principal component analysis (SSPCA), specific variable loadings in principal components are set to zero to pinpoint these significant locations. The effectiveness of SSPCA is evaluated through experiments on the UK 77-bus system and compared with traditional principal component analysis, demonstrating similar outcomes in identifying critical points. The study emphasizes SSPCA's validity and potential through simulation and comparative analysis, showcasing its usefulness in aiding power operators/engineers in prioritizing critical locations for deploying advanced measurement systems such as Phasor Measurement Units (PMUs).

Channel identification without prior knowledge is crucial in communication systems [3]. The distributed subchannel matching (DSCM) algorithm is widely used for single-input multi-output (SIMO) system identification. However, its effectiveness diminishes when dealing with sparse channels. To address this limitation, this paper introduces sparse DSCM algorithms that leverage sparsity-enforcing regularization terms in the cost function. Through simulations, their proposed approach demonstrates enhanced performance in both convergence and accuracy for channel identification.

The rise in renewable energy adoption and the integration of distributed generators and electric vehicles have complicated fault diagnosis within distribution networks [4]. To address this challenge, advanced telecommunications technology and high-frequency sensors are being deployed for grid modernization. This study introduces a sparse matrix reconstruction technique utilizing Phasor Measurement Units (PMUs) to detect faults. By optimizing PMU placement and analyzing voltage differentials before and during faults, the proposed algorithm identifies faults using the least absolute shrinkage and selection operator (LASSO), enhancing fault detection in power systems.

Cross-platform development frameworks have gained a lot of attention because of their capability to simplify the process of developing mobile applications that can run consistently across diverse platforms. Xamarin, a predecessor to .NET MAUI, introduced this approach to developers to build native mobile applications using C# and .NET. One of the main target of .NET MAUI is to authorize user to implement as much as application logic and user interface layout [5].

However, as the urge for more powerful and feature-rich applications continuously expands, the necessity for advancements in cross-platform development tools also grows. This need prompted the development of .NET MAUI, which represents an enormous transformation from Xamarin while providing new capabilities and advancements. .NET MAUI takes advantage of the power of .NET 6 by introducing a unified project structure, simplified APIs, and more effective tooling to consolidate the development process.

.NET MAUI additionally considers essential development concerns, such as performance optimization and native platform integration. The need for performance optimization in cross-platform app development, particularly in terms of improving user experience and resource utilization. With functions like Hot Reload and platform-specific renderers, .NET MAUI strives to provide high-performance apps with native-like user interfaces across platforms [5].

II. METHODS

We have worked on or tested on) ReadDataFromCsvAsync(Stream fileStream), ReadDataFromCsvContent(string csvContent), HeatMapInputModel Class, MauiProgram Class, HandleValidSubmit(), CalculateChartDimensions(), GenerateAxisLabels(int min, int max, bool isVertical), GenerateAxisLabelsVertical(int min, int max, bool isVertical) . The following are examples of some of those techniques.

- *MauiProgram Class:*

The MauiProgram class is responsible for creating and configuring a Maui application instance. It sets up various components and services required for the

```
public static class MauiProgram
   {
      public static MauiApp CreateMauiApp()
      {
          //Creating MAUI app instance
         var builder = MauiApp.CreateBuilder();
         builder
            .UseMauiApp<App>()
            .ConfigureFonts(fonts =>
            {
               fonts.AddFont("OpenSans-Regular.ttf",
"OpenSansRegular");
            });

         builder.Services.AddMauiBlazorWebView();

builder.Services.AddSingleton<CsvDataService>();
         builder.Services.AddSingleton<IPath,
PathService>();

#if DEBUG

builder.Services.AddBlazorWebViewDeveloperTools();
         builder.Logging.AddDebug();
#endif

         return builder.Build();
      }
```

- *HeatMapInputModel Class:*

The HeatmapInputModel class serves as a structured representation of user input and configuration parameters required for generating a heatmap visualization.

```
public class HeatmapInputModel
{
         //setting and retrieving data from user inputs
   public IBrowserFile CsvFile { get; set; }
   public string CsvContent { get; set; }
   public bool UseFileInput { get; set; } = true;

    // if any required field is empty giving error message

   [Required(ErrorMessage = "Highlight touch is
required.")]
   public int HighlightTouch { get; set; }

   [Required(ErrorMessage = "Figure name is
required.")]
   public string FigureName { get; set; }

   [Required(ErrorMessage = "X Axis label is
required.")]
   public string XAxisTitle { get; set; }

   [Required(ErrorMessage = "Y Axis label is
required.")]
   public string YAxisTitle { get; set; }

   public int? MaxCycles { get; set; }
   public int? MinCell { get; set; }
   public int? MaxCell { get; set; }
   public int? MinTouch { get; set; }
   public int? MaxTouch { get; set; }
   public bool IsHorizontal { get; set; }
}
```

- *ReadDataFromCsvAsync(Stream fileStream) :*

This C# function reads data from a CSV file stream asynchronously. After parsing the CSV data, integers are extracted and arranged into sets. The list of sets, as well as the maximum and minimum values discovered in the CSV data, are returned in a tuple by the procedure. The method ReadDataFromCsvAsync indicates that it returns a Task that represents an asynchronous operation. The returned task holds a tuple containing a list of HashSet<int> (sets of integers), the maximum value found, and the minimum value found. If there are any valid integer values (values.Count > 0), a new HashSet<int> containing these values is created and added to dataSets. All individual integer values are added to allCells using AddRange(). After parsing all

data, the maximum and minimum values are calculated from allCells.

```
public async Task<(List<HashSet<int>>, int, int)> ReadDataFromCsvAsync(Stream fileStream)
 {
    try
    {
       var dataSets = new List<HashSet<int>>();
       var allCells = new List<int>();

       // Use StreamReader in asynchronous mode
       using (var reader = new StreamReader(fileStream))
       {
          string line;
          while ((line = await reader.ReadLineAsync()) != null)
          {
             var values = line.Split(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries)
                              .Select(v => v.Trim())
                              .Where(v => !string.IsNullOrWhiteSpace(v) && int.TryParse(v, out _))
                              .Select(int.Parse)
                              .ToList();

             if (values.Count > 0)
             {
                var cell = new HashSet<int>(values);
                dataSets.Add(cell);
                allCells.AddRange(cell);
             }
          }
       }
       var maxCell = allCells.Count > 0 ? allCells.Max() + 100 : 0;
       var minCell = allCells.Count > 0 ? allCells.Min() - 100 : 0;
       return (dataSets, maxCell, minCell);
    }
    catch (Exception ex)
    {

       throw;
    }
 }
```

- *ReadDataFromCsvContent(String csvContent):*

This method's objectives are to handle CSV data that is supplied as a string, extract integers, group them into sets, and compute the data's maximum and minimum values. The CSV content string is split into lines using the Split() method, ensuring that empty lines are ignored. For each line in the CSV content, the method first Splits the line by commas to separate individual values. Secondly, Trim whitespace from each value and filter out empty or non-integer values. Finally, Parses valid integer values into a list called values.

```
 public (List<HashSet<int>>, int, int) ReadDataFromCsvContent(string csvContent)
  {
     var dataSets = new List<HashSet<int>>();
     var allCells = new List<int>();

     // Split the input CSV content by lines
     var lines = csvContent.Split(new[] { '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);

     foreach (var line in lines)
     {
        var values = line.Split(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries)
                         .Select(v => v.Trim())
                         .Where(v => !string.IsNullOrWhiteSpace(v) && int.TryParse(v, out _))
                         .Select(int.Parse)
                         .ToList();

        if (values.Count > 0)
        {
           var cell = new HashSet<int>(values);
           dataSets.Add(cell);
           allCells.AddRange(cell);
        }
     }
     var maxCell = allCells.Count > 0 ? allCells.Max() + 100 : 0;
     var minCell = allCells.Count > 0 ? allCells.Min() - 100 : 0;
     return (dataSets, maxCell, minCell);
  }
}
```

- *HandleValidSubmit():*

The HandleValidSubmit method is responsible for processing user input after the form submission, validating the input, and preparing data for generating a heatmap chart. Firstly, The method first checks whether the user input is valid based on the selected input method (UseFileInput) and the presence of CSV data.If the user opted for file input (UseFileInput is true) and a CSV file is provided (CsvFile is not null), the method reads the CSV data from the uploaded file using the CsvDataService.ReadDataFromCsvAsync method.If the user opted for direct input (UseFileInput is false) and CSV content is provided (CsvContent is not

empty), the method reads the CSV data from the content using the CsvDataService.ReadDataFromCsvContent method.Secondly, The method assigns various properties and settings based on the user input, such as highlightTouch, figureName, xAxisTitle, yAxisTitle, isHorizontal, numTouches, and count.highlightTouch is adjusted by subtracting 1.The number of touches (numTouches) is determined based on the minimum of either the count of active cells or the maximum cycles provided by the user, with a default maximum value of 1000.Finally, CalculateChartDimensions and StateHasChanged is called to determine the dimensions of the heatmap chart based on the provided data and to notify the UI that the state has been updated and it needs to be re-rendered.

```
private async Task HandleValidSubmit()
{
        if (heatmapInputModel.UseFileInput &&
heatmapInputModel.CsvFile != null)
        {
                var maxFileSize = 1024 * 1024;
                var stream =
heatmapInputModel.CsvFile.OpenReadStream(maxFile
Size);
                var result = await
CsvDataService.ReadDataFromCsvAsync(stream);
                activeCellsColumn = result.Item1;
        }
        else if (!heatmapInputModel.UseFileInput &&
!string.IsNullOrWhiteSpace(heatmapInputModel.CsvCo
ntent))
        {
                var result =
CsvDataService.ReadDataFromCsvContent(heatmapIn
putModel.CsvContent);
                activeCellsColumn = result.Item1;
        }
        highlightTouch =
heatmapInputModel.HighlightTouch - 1;
        figureName =
heatmapInputModel.FigureName;
        xAxisTitle = heatmapInputModel.XAxisTitle;
        yAxisTitle = heatmapInputModel.YAxisTitle;
        isHorizontal =
heatmapInputModel.IsHorizontal;
        numTouches =
Math.Min(activeCellsColumn.Count,
heatmapInputModel.MaxCycles.HasValue ?

        heatmapInputModel.MaxCycles.Value : 1000);
        count = 0;
        //Swapping height and width of a cell based on
orientation
        if (isHorizontal)
        {
                cellWidth = originalCellHeight;
                cellHeight = originalCellWidth;
        }
        else
        {
                cellWidth = originalCellWidth;
                cellHeight = originalCellHeight;
        }
        //Dynamically calculating charts dimention
        CalculateChartDimensions();
        StateHasChanged();

}
```

- *CalculateChartDimensions()*

The CalculateChartDimensions method is responsible for determining the dimensions (width and height) of the heatmap chart based on the data provided and the user's configuration. It Initializes minCell with the maximum possible integer value and maxCell with the minimum possible integer value to ensure proper calculation. Then Iterates through each column in the activeCellsColumn. For each column, calculate the minimum and maximum cell values. After that updates minCell and maxCell accordingly to determine the overall range of cells present in the heatmap data. If no data is found (minCell is still int.MaxValue and maxCell is still int.MinValue), sets minCell and maxCell to 0 to avoid errors in subsequent calculations. In terms of minTouch and maxTouch we set the range of touches (minTouch and maxTouch) based on the number of touches provided by the user (numTouches). If the chart orientation is horizontal (isHorizontal is true) we calculate the width of the chart based on the range of cells (cellRangeSpan), cell width (cellWidth), cell padding (cellPadding), and additional padding (chartPadding), add an extra 50 units for visualization margin. Furthermore, we calculate the height of the chart based on the number of touches, cell height (cellHeight), cell padding, and additional padding. Adds an extra 200 units for visualization margin. If the chart orientation is vertical (isHorizontal is false), it calculates the height and width of the chart in a similar manner but swaps the roles of width and height.

```
//Dynamically calculating chart dimention based on data
provided
private void CalculateChartDimensions()
 {
   minCell = int.MaxValue;
   maxCell = int.MinValue;
   foreach (var column in activeCellsColumn)
   {
     if (column.Any())
     {
       int currentMin = column.Min();
       int currentMax = column.Max();

       if (currentMin < minCell)
         minCell = currentMin;
       if (currentMax > maxCell)
         maxCell = currentMax;
     }
   }
```

```
 if (minCell == int.MaxValue && maxCell == int.MinValue)
    {
       minCell = 0;
       maxCell = 0;
    }

    minTouch = 0;
    maxTouch = numTouches;

    if (isHorizontal)
    {
       int cellRangeSpan = activeCellsColumn.Max(col => col.Max()) + 1 - minCell;
       chartWidth = (cellRangeSpan * (cellWidth -2)) + chartPadding + 200;
       chartHeight = (numTouches * (cellHeight + cellPadding)) + chartPadding +50;
    }
    else
    {
       int cellRangeSpan = activeCellsColumn.Max(col => col.Max()) + 1 - minCell;
       chartHeight = (cellRangeSpan * (cellHeight - 2)) + chartPadding + 50;
       chartWidth = (numTouches * (cellWidth + cellPadding)) + chartPadding + 200;
    }
 }
```

- *GenerateAxisLabels(int min, int max, bool isVertical):*

The GenerateAxisLabels method in the project plays a critical role in dynamically generating axis labels for the heatmap chart. By taking into account the provided minimum and maximum values, it calculates a suitable step size to ensure even spacing of labels along the axis. Through an iterative process, it generates labels for each value within the specified range, converting integer values to strings for labeling purposes. The method also calculates the precise position of each label along the axis, considering whether the axis is vertical or horizontal. Additionally, it ensures that the last label is appropriately placed, especially in the case of a vertical axis. Overall, this method contributes significantly to the readability and interpretability of the heatmap chart, enhancing the user experience and facilitating accurate data visualization.

```
//Dynamically generate labels for axis when working with horizontal chart
private IEnumerable<(string label, int pos)> GenerateAxisLabels(int min, int max, bool isVertical)
{
        var labels = new List<(string label, int pos)>();
        if (min > 0 && min < 10)
        {
                min = 0;
        }
        int range = max - min;
        // Determine the magnitude of the range to set an appropriate step value.
        int magnitude = (int)Math.Pow(10, (int)Math.Log10(range) - 1);
        int step = (range / magnitude < 5) ? magnitude : magnitude * 5;
        string label;
        int pos;
        // Generate labels based on the calculated step value.
        for (int i = min; i <= max; i += step)
        {
                label = i.ToString();
                if (isVertical)
                {
        // Adjust position calculation for vertical orientation
        Pos = Convert.ToInt32(((float)(i - min) * (cellHeight - 2)));
                }
                else
                {
// Adjust position calculation for horizontal orientation
        pos = Convert.ToInt32(((float)(i - min) * ((cellWidth - 1) + cellPadding)) + chartPadding);
                }

                labels.Add((label, pos));
        }
        // Ensure the last label is always added at the end of the axis.
        if (!labels.Any(l => l.label == max.ToString()))
        {
        if (isVertical)
                {
                        pos = Convert.ToInt32(((float)(max - min) * (cellHeight - 2)));
                        labels.Add((max.ToString(), pos));
                }
                else
                {
                        pos = Convert.ToInt32(((float)(max - min) * ((cellWidth - 1) + cellPadding)) + chartPadding);
                        labels.Add((max.ToString(), pos));    }
                return labels;
        }
        return labels;
}
```

The GenerateAxisLabelsVertical method is a crucial component of our project's heatmap charting functionality. This method dynamically generates axis labels for the vertical axis of the heatmap chart based on the provided minimum and maximum values. It employs a dynamic adjustment mechanism to determine label density, ensuring that labels are evenly spaced along the axis while considering the chart's orientation and dimensions. Using an iterative approach, it creates labels for every value within the defined range, transforming integer values into strings to serve as labels. Importantly, it calculates the precise position of each label along the axis, incorporating factors such as cell height and padding. The method effectively handles the addition of the maximum label to ensure comprehensive axis labeling. In summary, this approach substantially improves the clarity and comprehensibility of the heatmap chart, enriching user interaction and enabling precise data interpretation.

```
//Dynamically generate labels for axis when working with vertical chart
        private IEnumerable<(string label, int pos)> GenerateAxisLabelsVertical(int min, int max, bool isVertical)
        {
                var labels = new List<(string label, int pos)>();
                if (min > 0 && min < 10)
                {
                        min = 0;
                }
                int range = max - min;
                // Determine the magnitude of the range to set an appropriate step value.
                int magnitude = (int)Math.Pow(10, (int)Math.Log10(range) - 1);
                int step = (range / magnitude < 5) ? magnitude : magnitude * 5;
                string label;
                int pos;
                // Generate labels based on the calculated step value.
                for (int i = min; i <= max; i += step)
                {label = i.ToString();
                        if (isVertical)
                        {
        pos = Convert.ToInt32(((float)(i - min) * ((cellHeight - 1) + cellPadding)) + chartPadding) - 12;
                        }
                        else
                        {
pos = Convert.ToInt32(((float)(i - min) * (cellWidth - 2))) + 100;              }
      labels.Add((label, pos));
      }
```

```
// Ensure the last label is always added at the end of the axis.
                if (!labels.Any(l => l.label == max.ToString()))
                {
                        if (isVertical)
                        {
        pos = Convert.ToInt32(((float)(max - min) * ((cellHeight - 1) + cellPadding)) + chartPadding) - 12;
    labels.Add((max.ToString(), pos));
                        }
                        else
                        {
pos = Convert.ToInt32(((float)(max - min) * (cellWidth - 2))) + 120;
labels.Add((max.ToString(), pos));
                        }
                        return labels;
                }
                return labels;
        }
        #endregion
}
```

- DownloadSVG():

The DownloadSVG() method asynchronously retrieves SVG content. It then saves this content as an image file named "heatmap.png" and displays an alert confirming the completion of the download, along with the file name.

.

```
private async Task DownloadSVG()
{
        // Invoke the JavaScript function to get the SVG content
        var svgContent = await JSRuntime.InvokeAsync<string>("getSvgContent");

        var fileName = "heatmap.png";

        // Convert the SVG content to an image and save it
        await SaveSvgAsImageAsync(svgContent, fileName);

        await DisplayAlert("Download Complete", $"The image has been saved to {fileName}.", "OK");
}
```

- SaveSvgAsImageAsync(string svgContent, string filename):

The SaveSvgAsImageAsync() method asynchronously converts SVG content into a PNG image file. It first parses the SVG content and creates a bitmap to render the SVG onto. Then, it encodes the bitmap as a PNG image and determines the save path within the cache directory. Finally, it writes the image data to a file at the specified path and notifies the user about the completion of the operation, including the file path.

```
public async Task SaveSvgAsImageAsync(string svgContent, string filename)
{
        // Parse the SVG content
        var svg = new SKSvg();
        svg.FromSvg(svgContent);

        // Create a bitmap to render the SVG onto
        var bitmap = new SKBitmap((int)svg.Picture.CullRect.Width, (int)svg.Picture.CullRect.Height);
        using var canvas = new SKCanvas(bitmap);
        canvas.Clear(SKColors.White);
        canvas.DrawPicture(svg.Picture);
        canvas.Flush();

        // Encode the bitmap as a PNG
        using var image = SKImage.FromBitmap(bitmap);
        using var data = image.Encode(SKEncodedImageFormat.Png, 80);

        // Get the save path
        string folderPath = FileSystem.CacheDirectory;
        string filePath = Path.Combine(folderPath, filename);

        // Write the image data to a file
        using var stream = File.OpenWrite(filePath);
        data.SaveTo(stream);

        await NotifyUser(filePath);

}
```

III. RESULTS

In the Results section, the findings of the implementation of the SDR representation in the MAUI application are presented.

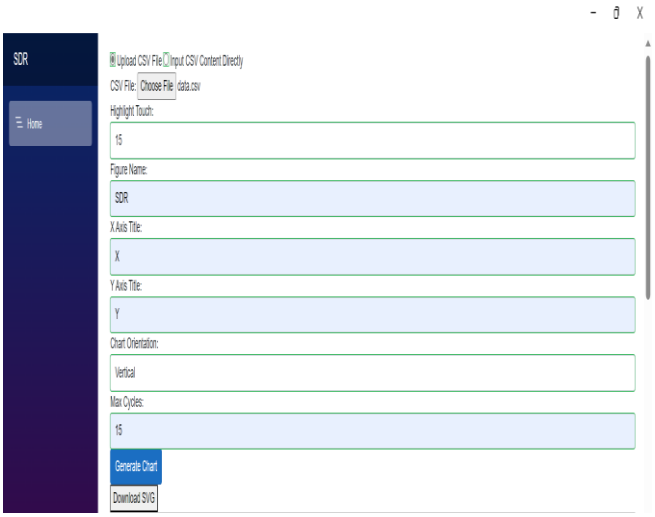

Figure 1: Input data from local device

Figure 1 illustrates how a user can select an input data file from his local device to generate the SDR.

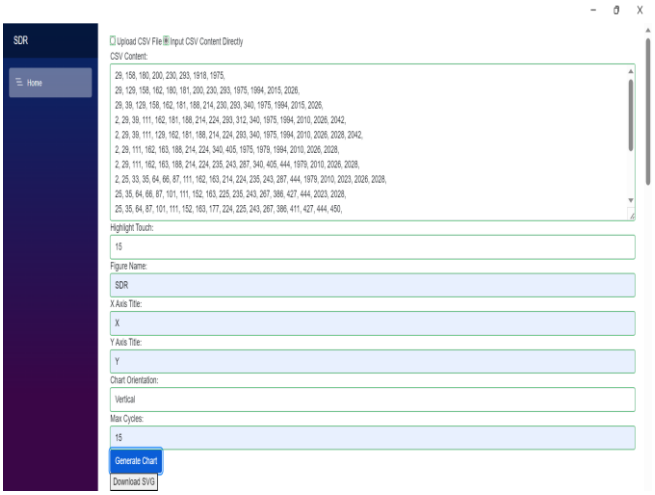

Figure 2: Pasting data manually for SDR representation

Figure 2 represents that a user can give input data manually into text field for generating the output.

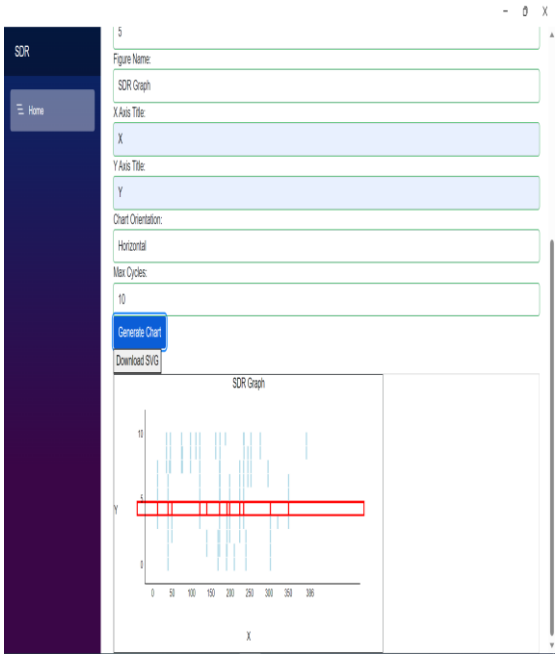

Figure 3: Horizontal Output

Figure 3 represents the horizontal output of the SDR

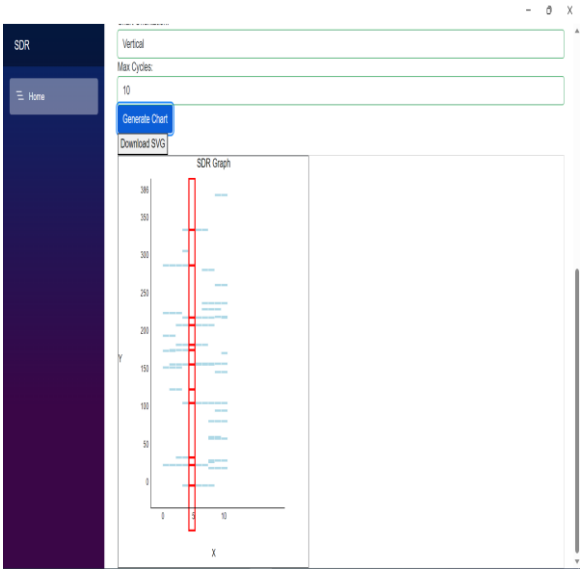

Figure 4: Vertical Output

Figure 4 represents the vertical output of the SDR.

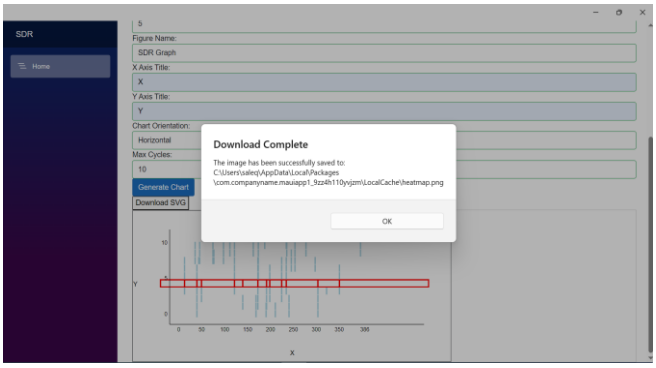

Figure 5: User Downloading Generated SDR chart

Figure 5 shows how a user can download the generated image.

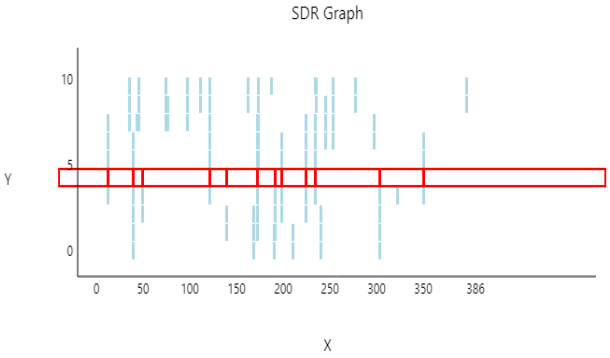

Figure 6: Downloaded image

Figure 6 represents the downloaded image that the user downloaded.

IV. CONCLUSION

In this study, we successfully developed an MAUI application using C# that represents an SDR both horizontally and vertically, in alignment with the requirements outlined by our professor. Through our implementation, we have demonstrated the practical application of theoretical concepts learned in our coursework, particularly in the realm of user interface design and software development. Our main findings indicate that utilizing C# and MAUI for developing SDR representations offers a versatile and efficient platform for visualizing complex data structures and functionalities. By aligning our implementation closely with the Python script provided by our professor, we were able to ensure consistency and relevance to the course objectives. Our project contributes to the existing body of research by showcasing the adaptability of modern programming languages and frameworks in educational settings. It emphasizes the importance of hands-on experience in reinforcing theoretical knowledge and fostering practical skills in software development. However, despite our achievements, our study has certain limitations. Firstly, our focus was primarily on meeting the requirements specified by our professor, which may have restricted the scope of exploration and innovation. Additionally, the complexity of SDR systems warrants further investigation into advanced features and functionalities that could not be fully addressed within the confines of this project. These limitations serve as catalysts for future research endeavors. Future studies could delve deeper into optimizing the performance of MAUI applications for SDR representations, exploring additional features such as signal processing algorithms, and extending compatibility to other platforms beyond the scope of our current implementation.

REFERENCES

[1] Mallick, A. and Joshi, G. (2019) 'Rateless codes for distributed computations with sparse compressed matrices', 2019 IEEE International Symposium on Information Theory (ISIT) [Preprint]. doi:10.1109/isit.2019.8849306.

[2] Zheng, P. et al. (2017) 'Using of sparse principal component analysis for identifying critical locations of the Active Distribution System', 2017 36th Chinese Control Conference (CCC) [Preprint]. doi:10.23919/chicc.2017.8029066. H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee, "RecentAdvances in Recurrent Neural Networks," pp. 1–21, 2017.

[3] Liu, Y., Liu, H. and Li, C. (2016) 'Distributed blind identification of sparse channels in sensor networks', 2016 35th Chinese Control Conference (CCC) [Preprint]. doi:10.1109/chicc.2016.7554150. A. Prayitno and S. Suyanto, "Segment Repetition Based on High Amplitude to Enhance a Speech Emotion Recognition," Procedia Comput. Sci., vol. 157, pp. 420–426, 2019.

[4] Kumar, Y. et al. (2023) 'Fault detection and localization in distribution system using sparse matrix reconstruction', 2023 International Conference on Computer, Electronics & Electrical Engineering & their Applications (IC2E3) [Preprint]. doi:10.1109/ic2e357697.2023.10262446.

[5] Davidbritch (2024) What is .net maui? - .net maui, .NET MAUI | Microsoft Learn. Available at: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-8.0 (Accessed: 29AD).