Implement the SDR representation in the MAUI application

Anh Tong Ngoc Minh   
[anh.tongngocminh@stud.fra-uas.de](mailto:anh.tongngocminh@stud.fra-uas.de)

Son Pham Tien  
[son.phamtien@stud.fra-uas.de](mailto:son.phamtien@stud.fra-uas.de)

*Abstract*— Due to the drastic demand of users in problems involving real-time solutions, Software developers have options for using sufficient frameworks. In the specification of the app user, a cross-platform framework, .NET Multi-platform App User Interface (.NET MAUI), is a good choice for developers to implement their models compatibly on multiple devices with less work. Besides, a separation in MAUI structure of visualization and functions provides better keeping up with the app development. This paper shows the creation of a .NET MAUI app to interact with users via User Interface (UI). The structure of vital iterations in generating the MAUI app is specified. The purpose is to replace the current Sparse Distributed Representation (SDR) visualization with a simpler and more efficient tool, the new library for SDR drawing implemented using Maui.Graphics.

Keywords—MAUI, Maui.Graphics, User Interface (UI), cross-platform APIs, Data Binding, XAML, Model-View-ViewModel (MVVM), Sparse Distributed Representation (SDR)

# Introduction

Visualization plays a vital role in checking and monitoring the project. Developers could point out any error in the system and correct it. Sparse Distributed Representation (SDR) visualization in this project is used to keep track of active cells, which is fundamental to how **the Hierarchical Temporal Memory (HTM) model represents information** [1]. SDR is essential for gaining insights of the underlying features and characteristics of the processed data.

SDR is a foundational concept in machine learning and artificial intelligence. An SDR is a binary representation of data characterized by its sparsity, where only an insignificant percentage of its bits are active at any given time. Inspired by neuroscience, SDRs mimic encoding information in the human brain, offering a powerful framework for representing and processing complex data patterns. The concept of active columns further enriches the SDR framework, as it comprises an array containing the information of active column indicators. These active columns play a crucial role in encoding and interpreting data, providing insights into relevant features and patterns within the dataset [2].

The concept of Active Columns within SDR is understood as how SDRs function. SDR consists of numerous bits, in which an insignificant fraction are active (set to 1) while the majority remain inactive (set to 0) at any given time. This activation pattern mirrors the behavior of neurons in the human brain, where a 1 represents an active neuron and a 0 signifies an inactive one. Crucially, each bit within an SDR holds semantic meaning, and the set of active bits in a representation collectively encode the semantic properties of the information being represented [1].

The flexibility provided by .NET MAUI allows developers to design intuitive interfaces that seamlessly integrate with underlying data structures. With features such as data binding, Extensible Application Markup Language (XAML), and Model-View-ViewModel (MVVM) architecture, developers can create interactive interfaces that enable users to work with active cell columns.

The paper describes the implementation of SDR representation visualization with equivalent required functions, using the Maui.Graphic graphics canvas. This approach offers users an intuitive and efficient means of working with SDR visualization, enhancing their experience and interaction with the MAUI application.

# Methods

This section describes the vital interfaces used in the .NET MAUI applications, involving the User Interface (UI) design, model structure for logic implementation, and SDR representation function with *Maui.GraphicsView*, and local accessing for file picking.

## User Interface design

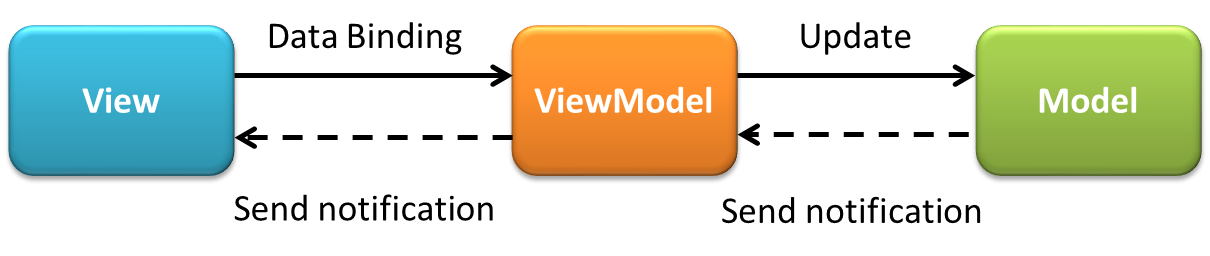
UI design is used to take input SDR parameters from the user. Utilizing the XAML crafts the UI by featuring input fields, representing images, sliders, and buttons. This design strategy allows users to define SDR parameters directly without the complexities of traditional command-line interfaces using Python.

The advantage of the design lies in the ability to create a responsive and easily navigable interface across various devices and orientations. XAML’s structure facilitates a more organized and visually coherent UI development process, aligning well with the MVVM architecture. This ensures a correlated connection between the UI and the underlying data models through data bindings [3].

Moreover, XAML's syntax is inherently more concise and readable compared to traditional coding methods. It mirrors the parent-child hierarchy of UI objects, enhancing the visual clarity of the interface structure.

## Using MVVM as structure for logic implementation

The properties of UI elements come with behind-logic functionality. As applications evolve, they may encounter challenges related to maintenance, such as intricate connections between UI controls and underlying logic, which hinder UI modifications and complicate unit testing.



*Figure 1: General structure of MVVM model*

The MVVM pattern offers a solution by separating an application's business and presentation logic from its UI elements. This separation enhances development by facilitating easier testing, and maintenance. Additionally, it promotes code reusability and fosters collaboration between developers and UI designers.

The three key components of MVVM are the model managing data, the view rendering the UI, and the view model serving as an intermediary between the model and the view. Fig.1 shows the relationships between the three components.

The properties of UI elements come with behind-logic functionality. The **View** holds the outline of the arrangement, design, and visual presentation of the content visible to the user on the screen. Each View is created using XAML, accompanied by minimal code-behind that refrains from integrating business logic.

The **ViewModel** sets up properties and commands where View can connect, allowing data binding. It informs the View about any changes in state using notification events. While the View Model determines what functionality the UI will offer based on the properties and commands it provides. The View decides how to display this functionality. Additionally, the view model manages the interactions between the View and any necessary Model class.

The **Model** is a class, having no visual representation, but is used to hold the data. As a result, the Model is seen as a reflection of the application's domain Model, which typically encompasses the data Model, and various business rules and validation logic.

## SDR reprentation function with Maui.GraphicsView

The GraphicsView, serves as a canvas for rendering 2D graphics, utilizing Microsoft.Maui.Graphics namespace types. Defined within this framework is the Drawable property, using an IDrawable interface, which delineates the rendered content. This particular property is supported by a BindableProperty, enabling it to become a focal point for data binding and styling processes. The general implementation of *GraphicsView()* in an XAML file is shown in Listing 1. *HeightRequest* and *WidthRequest* in Listing 1 are defined with constant values, while those are auto-adjusted in this project.

<ContentPage.Resources>

<drawable:GraphicsDrawable x:Key="drawable" />

</ContentPage.Resources>

<VerticalStackLayout>

<GraphicsView Drawable="{StaticResource drawable}"

HeightRequest="300"

WidthRequest="400" />

</VerticalStackLayout>

*Listing 1: The XAML code sniper of GraphicsView() in the UI*

## Accessing files in local device

The FilePicker() class in the .NET MAUI IFilePicker interface represents a significant leap forward in enhancing user experience by simplifying access and selection of files from the device's storage. The class bridges the gap between the application and the device's native file management capabilities, offering an intuitive interface for users across all supported platforms, including iOS, Android, macOS, and Windows [4].

Because MAUI-based applications in this project focus on data representation, such as those visualizing SDRs, FilePicker() enables users to import the necessary data files for visualization. Therefore, it appears as a more interactive experience for users. The ease with which users can select and upload files directly impacts the effectiveness of the data visualization process.

# Implementation

## UI implementation

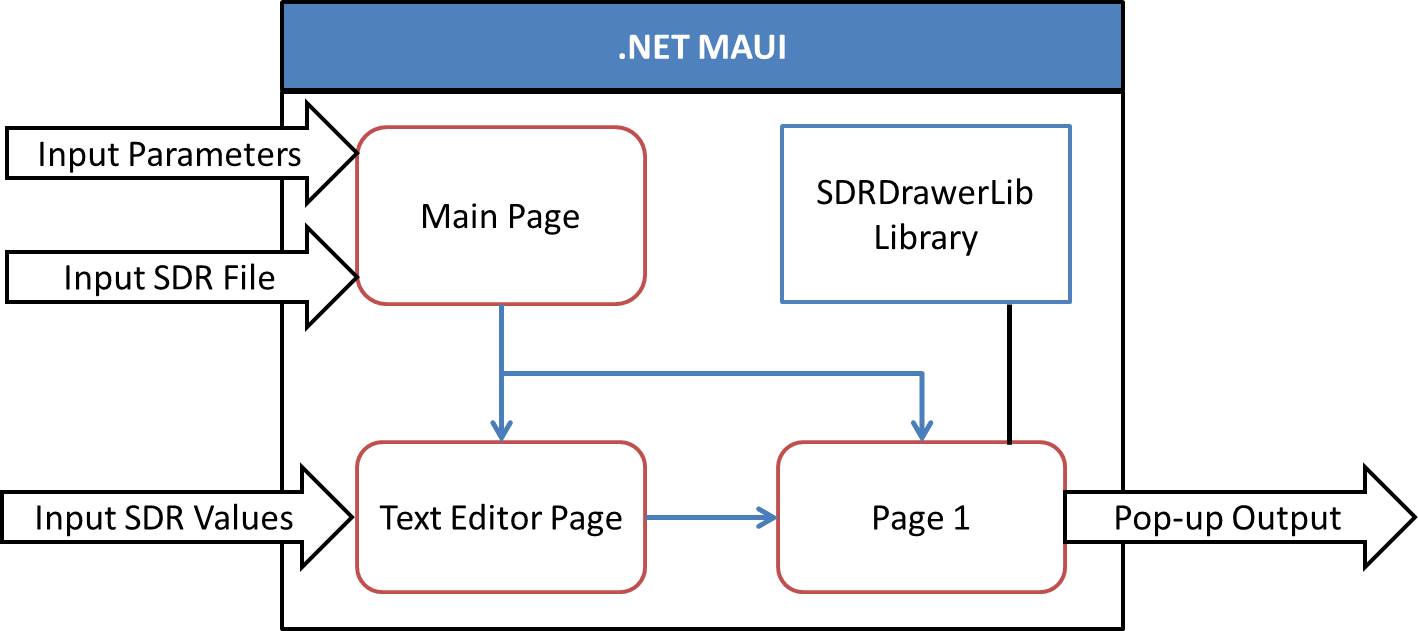
UI design is used to take input SDR parameters from the user. Utilizing the XAML crafts the UI by featuring input fields, representing images, sliders, and buttons. This design strategy allows users to define SDR parameters directly without the complexities of traditional command-line interfaces using Python.

The advantage of the design lies in the ability to create a responsive and easily navigable interface across various devices and orientations. XAML’s structure facilitates a more organized and visually coherent UI development process, aligning well with the MVVM architecture. This ensures a correlated connection between the UI and the underlying data models through data bindings [3].

Moreover, XAML's syntax is inherently more concise and readable compared to traditional coding methods. It mirrors the parent-child hierarchy of UI objects, enhancing the visual clarity of the interface structure.

## Drawing library implementation

The properties of UI elements come with behind-logic functionality. As applications evolve, they may encounter challenges related to maintenance,



*Figure 2: General structure of MAUI implementation*

## Logic implementation

fdfgdf, using an IDrawable interface, which delineates the rendered content. This particular property is supported by a BindableProperty, enabling it to become a focal point for data binding and styling processes. The general implementation of *GraphicsView()* in, using an IDrawable interface, which delineates the rendered content. This particular property is supported by a BindableProperty, enabling it to become a focal point for data binding and styling processes. The general implementation of *GraphicsView()* in

# Results and evaluation

fsdfdsdfsdf

1. Comparison characteristic within 3 cases

|  |  |  |  |
| --- | --- | --- | --- |
| **Variables** | **Case 1** | **Case 2** | **Case 3** |
| Input parameters | Table entry | Table entry | Table entry |
| Input SDR values | File | File | Keyboard |
| SDR columns | <30 | >1000 | 8 |
| Screen fit | Fit | Scroll view | Fit |
| Saved figure | Good quality | Bad quality | Good quality |

In this study, the potential of .NET MAUI in enhancing visualization and interaction capabilities of SDR applications is provided. Inspired by neuroscience principles, SDRs offer a powerful framework for representing complex data patterns, with active columns playing a crucial role in encoding and interpreting data. Leveraging the flexibility and cross-platform capabilities of .NET MAUI, user interaction is upgraded and facilitated ease of use in SDR applications.

Through the project, a user-friendly interface is successfully implemented for inputting SDR parameters, mirroring the functionality of traditional command-line interfaces and eliminating the need for manual input of file paths via command-line arguments. The .NET MAUI and XAML integration allowed us to design responsive and easily navigable interfaces across various devices and orientations, aligning well with the MVVM architecture.

Introducing features such as the File Picker System in .NET MAUI has significantly enhanced the user experience by simplifying the process of accessing and selecting files, facilitating a more interactive user experience. The additional Text Edit feature implementation empowers users to customize and input SDR values directly, enhancing flexibility and control.

Finally, users can save the output of SDR representations directly to their desktop as image files, enhancing the utility of the application and enabling further analysis or dissemination of visualized SDR representations.

# Conclusion

In conclusion, .NET MAUI is an integrated tool in app generation. With a combination of supported interfaces in .NET MAUI, a friendly user application is generated and implemented within the project. The app allows users to have one more option in inputting the SDR values and configuring the visual-defined parameters. The input local content is dynamic, so it is possible to implement the app on any device. The app is concerned about the user's input parameters, files, or entered SDR values, not the directory or dependent connection to any project; therefore, it can be integrated with distinguished projects having various demand.

The integration of .NET MAUI in SDR applications holds tremendous potential for enhancing user interaction, accessibility, and usability, and enhancing user satisfaction in software applications. Further research and development in this project promise to unlock even greater possibilities for leveraging SDR visualization.

##### Acknowledgment

We would like to thank Dr. Dobric for his active support and assistance in the direction and model configurations. We are thankful to our tutor for helping us solve encountered problems.

##### References

[1] S. Ahmad and J. Hawkins, ‘Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory’. arXiv, Mar. 25, 2015. doi: 10.48550/arXiv.1503.07469.

[2] ‘Sparse Distributed Representation and Hierarchy: Keys to Scalable Machine Intelligence’. Accessed: Mar. 29, 2024. [Online]. Available: https://apps.dtic.mil/sti/citations/tr/AD1006958

[3] davidbritch, ‘XAML - .NET MAUI’. Accessed: Mar. 29, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/xaml/?view=net-maui-8.0

[4] davidbritch, ‘File picker - .NET MAUI’. Accessed: Mar. 29, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/storage/file-picker?view=net-maui-8.0