{Citation}I. INTRO (HEADING 1)

This Section should be focused on describing your approach. You can use references from other source.

Sparse Distributed Representation (SDR) stands as a foundational concept in the realm of machine learning and artificial intelligence. An SDR is a binary representation of data characterized by its sparsity, where only a small percentage of its bits are active at any given time. Inspired by the principles of neuroscience, SDRs mimic the encoding of 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.

The concept of Active Columns within Sparse Distributed Representation (SDR) can be understood in the context of how SDRs function. SDRs consist of numerous bits, a small fraction of which are activated (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 [6].

Therefore, representing Active Columns within Sparse Distributed Representation (SDR) is essential for gaining insights into the underlying features and characteristics of the data being processed.

The flexibility provided by .NET MAUI empowers developers to design intuitive interfaces that seamlessly integrate with underlying data structures. With features such as data binding, XAML, and model-view-viewmodel (MVVM) architecture, developers can create interactive interfaces that enable users to interact with active cell columns effortlessly.

This paper introduces an implementation of Sparse Distributed Representation (SDR) with equivalent functionality to that of a Python file, using MAUI.Graphic. This approach offers users an intuitive and efficient means of working with SDR visualization, enhancing their experience and interaction with the MAUI application.

II. METHODS

**A. User Interface Design**

In our approach, we leverage .NET MAUI to design a user interface (UI) that significantly enhances user interaction for inputting Sparse Distributed Representation (SDR) parameters. By utilizing the Extensible Application Markup Language (XAML), we crafted a user-centric interface featuring intuitive input fields, sliders, and buttons. This design strategy allows users to efficiently define SDR parameters directly without delving into the complexities of traditional command-line interfaces using draw\_figure Python.

The advantage of employing .NET MAUI and XAML for our UI design lies in the ability to create a responsive and easily navigable interface across various devices and orientations. XAML's XML-based structure facilitates a more organized and visually coherent UI development process, aligning well with the Model-View-ViewModel (MVVM) architecture. This ensures a seamless connection between the UI and the underlying data models through data bindings [1].

Moreover, XAML's markup 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.

**B. Reprentation of SDR with Maui.GraphicView**

The GraphicsView, as part of the .NET Multi-platform App UI (.NET MAUI), serves as a canvas for the rendering of 2D graphics, utilizing the Microsoft.Maui.Graphics namespace types. Defined within this framework is the Drawable property, of the IDrawable type, which delineates the content to be rendered. This particular property is supported by a BindableProperty, enabling it to become a focal point for data binding and styling processes[2].

namespace MyMauiApp

{

public class GraphicsDrawable : IDrawable

{

public void Draw(ICanvas canvas, RectF dirtyRect)

{

// Drawing code goes here

}

}

}

*Listing1: Implementing drawing code in MAUI.Graphics*

The detail Draw method is conducted in following folder [3]  
The Draw function requires two parameters: ICanvas and RectF. ICanvas represents the drawing surface for rendering graphical elements, while RectF stores details about the canvas's dimensions and position.

In XAML, IDrawable objects can be defined as resources and referenced by a GraphicsView using their unique key assigned as values of the Drawable property. This enables efficient management and reuse of graphical content throughout the application, facilitating modular design and improving maintainability.

By employing this approach, developers can seamlessly integrate complex graphical elements into user interfaces while ensuring consistency and scalability in design implementation.

<ContentPage.Resources>

<local:Page1ViewModel x:Key="drawable"/>

</ContentPage.Resources>

<ScrollView>

<GraphicsView Drawable="{StaticResource drawable}"

Grid.Column="1"

Grid.Row="0"

x:Name="DrawableView"/>

</ScrollView>

*Listing 2: IDrawable object can be declared as a resource and then consumed by a GraphicsView in XAML code.*

In the listing 2, the **IDrawable** object is declared as a resource and then consumed by a GraphicsView by specifying its key “drawable” as the value of the Drawable property

**C.MVVM (Model-View-Viewmodel)**

In the .NET MAUI development process, designers typically utilize XAML to create the user interface, followed by implementing code-behind functionalities to interact with it. 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.

The model-view-viewmodel (MVVM) architectural 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, maintenance, and evolution of the application. Additionally, it promotes code reusability and fosters collaboration between developers and UI designers[4].

MVVM comprises three key components: the model, responsible for managing data; the view, responsible for rendering the UI; and the view model, serving as an intermediary between the model and the view. The following diagram shows the relationships between the three components.



Figure 1: The diagram illustrates MVVM Architecture in MAUI application[https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm]

**1. View**

The view holds the task of outlining the arrangement, design, and visual presentation of the content visible to the user on the screen. It is preferable for each view to be created using XAML, accompanied by minimal code-behind that refrain from integrating business logic.

**ViewModel**

The view model sets up properties and commands that the view can connect with, 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, it's the view that decides how to display this functionality. Additionally, the view model manages the interactions between the view and any necessary model classes.

**Model**

Model classes are types of classes that don't have a visual representation but are used to contain the data of the application. As a result, the model can be seen as a reflection of the application's domain model, which typically encompasses both a data model and various business rules and validation logic.

**D.File Picker System :**

The File Picker System in .NET MAUI represents a significant leap forward in enhancing user experience by simplifying the process of accessing and selecting files from the device's storage. This system is ingeniously designed to bridge the gap between the application and the device's native file management capabilities, offering a seamless and intuitive interface for users across all platforms supported by .NET MAUI, including iOS, Android, macOS, and Windows [5].

In the context of MAUI-based applications focused on data representation, such as those visualizing Sparse Distributed Representations (SDRs), the File Picker System plays a pivotal role. It enables users to effortlessly import the data files necessary for visualization, thereby facilitating a more interactive and user-friendly experience. The ease with which users can select and upload files directly impacts the effectiveness of the data visualization process, making the File Picker System a crucial component of the application's overall architecture and user interaction strategy.

**RESULTS**

The implementation of the user interface (UI) and the SDR representation successfully mirrors the functionality of the Python file. Through the UI, users can input parameters conveniently, enabling seamless interaction with the SDR representation. The UI design incorporates elements such as input fields, and buttons, ensuring user-friendly access to define SDR parameters. This interface allows users to effortlessly manipulate the SDR representation according to their requirements.

A screenshot of a computer

Description automatically generated

*Figure 2: The User Interface enables user input parameters.*

A vertical chart with numbers and text

Description automatically generated with medium confidence A screenshot of a graph

Description automatically generated

*Figure 3: Comparision SDR Represenation from Python file and MAUI Application*

As a result of implementing the MAUI (Multi-platform App UI) application, users can now conveniently generate SDR (Sparse Distributed Representation) representations without the need to rely on the command line interface in the Python file. The application enables users to effortlessly select the desired text file from their device's file system, eliminating the manual input of file paths via command line arguments. As seen in Figure 3, the SDR representations generated from the MAUI application closely resemble those generated by the Python file.

Besides choosing text file to visualize SDR value, users now have the added functionality of a Text Edit feature, allowing them to customize and input SDR values directly. This enhancement provides users with greater flexibility and control over the SDR representations generated by the application. By integrating this feature, users can fine-tune and experiment with different SDR configurations, tailoring them to specific needs or preferences. This expanded capability enhances the overall usability and versatility of the application, empowering users to interact with SDR representations in a more intuitive and dynamic manner. The Text Editor enable user input are shown in the figure below:

A screenshot of a computer

Description automatically generated

*Figure4: Text Editor enable user customization of SDR value.*

Finally, users are now able to save the output of the SDR Representation directly to their desktop as an image file. This feature enhances the utility of the application by providing users with the capability to preserve and share their generated SDR representations in a convenient and accessible format. With the ability to save the output as an image, users can easily incorporate the visualized SDR representations into presentations, documents, or other media for further analysis or dissemination.

1. **CONCLUSION**

In this study, we have explored the potential of .NET Multi-platform App User Interface (MAUI) in enhancing the visualization and interaction capabilities of Sparse Distributed Representation (SDR) applications. Inspired by the principles of neuroscience, 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, we aimed to streamline user interaction and facilitate ease of use in SDR applications.

Through our approach, we have successfully implemented a user-friendly interface 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 integration of .NET MAUI and XAML allowed us to design responsive and easily navigable interfaces across various devices and orientations, aligning well with the Model-View-ViewModel (MVVM) architecture.

The introduction of 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, thereby facilitating a more interactive and user-friendly experience. Additionally, the implementation of a Text Edit feature empowers users to customize and input SDR values directly, enhancing flexibility and control.

Finally, users now have the capability to 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.

In conclusion, 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 SDRs visualization.

REFERENCES

[1] <https://learn.microsoft.com/en-us/dotnet/maui/xaml/?view=net-maui-8.0>

[2] <https://learn.microsoft.com/en-us/dotnet/maui/user-interface/graphics/?view=net-maui-8.0>

[3] <https://github.com/tongngocminhanh/MAUI_App_SDR/blob/master/MySEProject/AppSDR/SdrDrawerLib/SdrDrawable.cs>

[4] https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm

[5] <https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/storage/file-picker?view=net-maui-8.0&tabs=windows>

[6] S. &. H. .. Ahmad, “ “Properties of sparse distributed representations and their application to hierarchical temporal memory.,”,” 2011. [Online]. Available: doi: 10.1371/journal.pone.0022149.