Justification for design choices

A) (2D vector vs 1D array pointer):

When designing a matrix class for a neural network library, where operations like addition, multiplication, and copying are common, choosing the right data structure is crucial for both performance and ease of use. Let's compare the two main approaches: using a 2D vector array vs. a 1D array with a pointer.

**1D Array with Pointer (Current Approach)**

**Structure:**

* A single contiguous block of memory to store the matrix elements.
* Elements are stored in row-major order or column-major order.

**Access Pattern:**

* Access element (i, j) with the formula data[i \* num\_columns + j] for row-major order.

**Advantages:**

* **Cache Efficiency:** Elements are stored contiguously in memory, which can lead to better cache performance, especially for large matrices.
* **Simplicity in Allocation:** Allocating a single contiguous block of memory is straightforward and efficient.
* **Ease of Linear Operations:** Operations like matrix multiplication can be efficiently implemented using loop unrolling and SIMD instructions.

**Disadvantages:**

* **Index Calculation:** Requires manual calculation of indices to access elements, which can be error-prone.
* **Less Intuitive:** Less intuitive to work with compared to a 2D vector when visualizing or debugging matrix operations.

**2D Vector Array**

**Structure:**

* A vector of vectors, where each inner vector represents a row (or column) of the matrix.

**Access Pattern:**

* Access element (i, j) with the notation data[i][j].

**Advantages:**

* **Ease of Use:** More intuitive and easier to work with, especially for beginners. Directly access elements with data[i][j].
* **Dynamic Resizing:** Each row (or column) can be resized independently, offering flexibility in certain applications.

**Disadvantages:**

* **Fragmented Memory:** Memory is not contiguous, which can lead to poor cache performance for large matrices.
* **Complex Allocation:** More complex memory allocation as each row (or column) is a separate allocation.

**Comparison for Common Operations**

**Matrix Addition**

* **1D Array:** Efficient with good cache locality.
* **2D Vector:** Intuitive but potentially slower due to non-contiguous memory access.

**Matrix Multiplication**

* **1D Array:** Can be optimized for performance with contiguous memory and better cache usage.
* **2D Vector:** Easier to implement but generally slower due to fragmented memory.

**Hadamard Product (Element-wise Multiplication)**

* **1D Array:** Efficient with direct access using calculated indices.
* **2D Vector:** Intuitive access with data[i][j] but potentially slower.

**Recommended Approach**

For a neural network library where performance is critical, the **1D array with a pointer** is generally the better choice. It offers better cache efficiency and simpler memory management, which can lead to significant performance improvements for large-scale matrix operations.

B) Encapsulation vs Modularity

C) Functions will only do one specific job

D) A result/destination matrix cannot be the same as the argument matrix. This si to prevent potential bugs.