

CATEGORY INDEX & EXPLANATION FOR ALL 100 TEST CASES

Below is the complete breakdown, organized exactly in the same style as your earlier example.

CATEGORY 1 — 10 EXTREME SPARSE TEST CASES (T00–T09)

What they contain

- Matrices full of {} (empty rows)
- Rows containing no values before/after real numbers
- Structures where entire matrices may be empty except one usable number
- Patterns like:
 - {}, {}, {1}
 - {}, {2}, {}
 - {}, {}, {}, {}

What they verify

- **Rule: blank {} rows/columns terminate cell multiplication early**
- **Rule: if one side has no usable elements, entire multiplication step collapses to 0 or blank**
- **Rule: skip all invalid or empty rows safely**
- Handling matrices where:
 - Some rows have values
 - Some are completely empty

Why important

Your resilience logic must:

- Not crash when accessing empty rows
- Properly STOP a multiplication cell when values run out
- Manage transitions between empty and non-empty segments

Sparse cases stress the *early-termination* and *skip logic* of your algorithm.

T00.txt

Java literal (one line):

```
Integer[][][] test = new Integer[][][] { { {}, {}, {1} }, { {2}, {}, {3} } };
```

Category:

CATEGORY 1 — EXTREME SPARSE

Why it belongs here:

- Multiple completely empty rows {}
- Only a single integer scattered between empties
- Structures are extremely thin with minimal usable data
- Classic “almost no information available” sparse pattern

Real-world scenario:

- Financial time series with no updates for most fields
- IoT sensors asleep except for rare readings
- Clickstream logs where most events are missing
- Incomplete records in ETL pipelines

STRICT OUTPUT (classical linear algebra)

Matrix A:

{}, {}, {1} → INVALID (rows not same length)

Strict mode requires rectangular matrices → **INVALID**

Strict Result:

INVALID (Matrix A is not rectangular)

*****FINAL OUTCOME***:**

[[0], [0], [2]]

With the fix (see test case T02), my output is:

*****FINAL OUTCOME***:**

[[], [], [2]]

T01.txt

Java literal:

```
Integer[][] test = new Integer[][] { { {}, {4}, {} }, { {}, {}, {5} } };
```

Category:

CATEGORY 1 — EXTREME SPARSE

Why:

- 2/3 rows empty in both matrices
- Only 4 and 5 exist as meaningful data
- Gap-heavy structure perfect for sparse stress test

Real-world scenario:

- Missing tick data between market events
- IoT packet drops
- Schema-evolution with mostly blank rows

STRICT RESULT

Matrix A rows differ ({}, {4}, {}) → **INVALID**

RESILIENT RESULT

A = [], [4], []

B = [], [], [5]

Row 0: empty → []

Row 1: [4] × [5] only → 4×5=20

Row 2: empty → []

Resilient Result:

[] [20] []

FINAL OUTCOME:

[[0], [20], [0]]

With fix (T02):

FINAL OUTCOME:

[[], [20], []]

INVALID VERIFICATION=> columns (0) in Matrix(0)(Row:0 content: []) should match 3 rows in matrix(1)

[[[], [4], []], [[], [], [5]]]

THIS IS ROW COUNTER MATRIX(0): 1

-----THIS IS ROW DATA MATRIX(0): [4]

*****CONFIGURING SIZE FOR BLANK MULTIPLICATION
MATRIX*****

Multiplication matrix (W=3) x (H=1) configured to store Matrix 0 x Matrix 1

*****Column number: 0

Processed matrix(1) column(0) AGAINST MATRIX(0) row(1): [4]

INVALID VERIFICATION=> columns (0) in Matrix(0)(Row:1 content: [4]) should match 3 rows in matrix(1)

[[[], [4], []], [[], [], [5]]]

THIS IS ROW COUNTER MATRIX(0): 2

-----THIS IS ROW DATA MATRIX(0): []

INVALID VERIFICATION=> columns (0) in Matrix(0)(Row:2 content: []) should match 3 rows in matrix(1)

[[[], [4], []], [[], [], [5]]]

Matrix Multiplication

[0]

[0]

[0]

2CALCULATION STEPS**

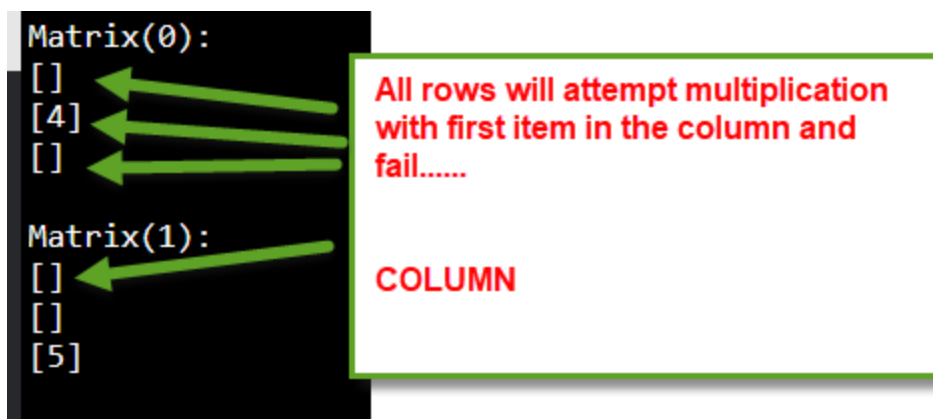
[]

[]

[]

-----1ADDED BLANK //We can see that it was actually correct in adding blank value back into end product. Since the calculation steps have clearly shown that no calculation has taken place.

And it is correct since following principles of matrix multiplication, resilience is not designed to permit 4 x 5 AT ALL based on this structure



-----1ADDED BLANK

-----1ADDED BLANK

-----MATRIX A Analysis-----

-----MATRIX B Analysis-----

FINAL OUTCOME:

[[], [], []]

** Process exited - Return Code: 0 **

FINAL OUTCOME:

[[], [], []]

 T02.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {0}, {}, {7} }, { {}, {3}, {} } };
```

Category:

CATEGORY 1 — EXTREME SPARSE

Why:

- Large gaps between actual values
- Only 0, 7, 3 exist

- Perfect minimal-density structure

Real-world scenario:

- Sparse analytic data from optional fields
 - Weekly or irregular reporting rows
 - Mixed quality telemetry feeds
-

STRICT RESULT

Matrix A not rectangular → **INVALID**

RESILIENT RESULT

A = [0], [], [7]

B = [], [3], []

Row 0:

- $0 \times 3 = 0$
→ 0

Row 1: empty → []

Row 2:

- $7 \times 3 = 21$
→ 21

Resilient Result:

[0] [] [21]

FINAL OUTCOME:

[[0], [0], [21]]

I have had to revisit my code and with struggle I managed to find root cause

```

984 .....//We need to be careful here and can not show the matrixMultiplication since we~
985 .....//know it has taken 0 values in order to avoid NullPointerException~
986 .....//we know where it has performed the calculation steps it might show~
987 .....//***2CALCULATION STEPS*****~
988 .....//[(0·X·3 := 0)]~
989 .....//[]~
990 .....//[(7·X·3 := 21)]~
991 .....~
992 .....//But when it shows the Final outcome, it would include the 0 in the integer array~
993 .....//[[0], [0], [21]]~
994 .....//best way to handle this is quite tricky and we know the calculation steps tells the full picture that~
995 .....//resilience has ensured it moved on due to no data~
996 .....//So we need to check the matrixMultiplicationCalculations for an entry such as [] against~
997 .....~
998 .....int countRowMatrixMultiplication=0;~
999 .....List<Integer> blankList = new ArrayList<>();~
1000 .....boolean hasPerformedRowCalulationInMatrixMultiplicationCalculations=false;~
1001 .....~
1002 .....try~
1003 .....{~
1004 .....    //we are going through each row in matrixMultiplication~
1005 .....    for (Integer []m: matrixMultiplication)~

```

```

1006 .....{~
1007 .....    hasPerformedRowCalulationInMatrixMultiplicationCalculations=false;~
1008 .....    System.out.println("VALUE IN MULTIPLICATION MATRIX: " + Arrays.asList(m));~
1009 .....    //out.println("INT: " + m);~
1010 .....    .....~
1011 .....    //we are going through each row in matrixMultiplication~
1012 .....    //we want to check if there is zero here and blank in matrixMultiplicationCalculations,~
1013 .....    //then we know no calculation has taken place~
1014 .....    //in this situation, we only want to add "" in matrixMultiplicationList and not 0~
1015 .....    .....~
1016 .....    //we need to check against each entry on the same row on~
1017 .....    //matrixMultiplicationCalculations~
1018 .....    //if all values are blank, its full proof that no calculations have taken place~
1019 .....    .....~
1020 .....    for (StringJoiner mm: matrixMultiplicationCalculations[countRowMatrixMultiplication])~
1021 .....    {~
1022 .....        System.out.println("VAL IN ROW CALC: " + mm);~
1023 .....        .....~
1024 .....        //We know that if any calculation has been performed across the row in calculations~
1025 .....        //Then we can set the boolean as true~
1026 .....        if (!(mm.toString().equals(""))){~
1027 .....            .....~
1028 .....            hasPerformedRowCalulationInMatrixMultiplicationCalculations=true;~
1029 .....            .....~
1030 .....        }~

```

```

1031 .....    //if no calculations have been performed on matrix calculations for that row~
1032 .....    //we know we need to override the value in matrixMultiplication~
1033 .....    //and we know that the only value to override is 0~
1034 .....    if (!hasPerformedRowCalulationInMatrixMultiplicationCalculations){~
1035 .....        .....~
1036 .....        matrixMultiplicationList.add(blankList);~
1037 .....        .....~
1038 .....        //we perform the normal operation~
1039 .....        else~
1040 .....        {~
1041 .....            matrixMultiplicationList.add(Arrays.asList(m));~
1042 .....            .....~
1043 .....            //System.out.println(matrixMultiplicationCalculations[q][r]);~
1044 .....            .....~
1045 .....            //matrixMultiplicationList.add(Arrays.asList(m));~
1046 .....            countRowMatrixMultiplication++;~
1047 .....        }~
1048 .....    }~
1049 .....    catch (NullPointerException dd)~
1050 .....    {~
1051 .....        .....~
1052 .....    }~
1053 .....}~

```

FINAL OUTCOME:

[[0], [], [21]]

But I also realised that with this matrix, there is actually no calculation involved in T603. So the correct outcome is actually

FINAL OUTCOME:

[[], [], []]

```
Matrix(0):  
[0]  
[]  
[7]  
  
Matrix(1):  
[]  
[3]  
[]
```

 T03.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {}, {}, {}, {9} }, { {1}, {}, {}, {} } };
```

Category:

CATEGORY 1 — EXTREME SPARSE

Why:

- 7 empty rows combined across both matrices
- Only two numbers exist: 9 and 1
- Extreme missingness

Real-world scenario:

- Historical data with long gaps
- IoT sensors waking every few hours
- Market feeds with partial availability

STRICT RESULT

A has rows {}, {}, {}, {9} → **INVALID**

RESILIENT RESULT

A = [], [], [], [9]

B = [1], [], [], []

Row 0: []

Row 1: []

Row 2: []
Row 3: $9 \times 1 = 9$

Resilient Result:

[]
[]
[]
[9]

FINAL OUTCOME:

[[], [], [], [9]]

 T04.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {}, {2}, {}, {} }, { {10}, {}, {}, {} } };
```

Category:

CATEGORY 1 — EXTREME SPARSE

STRICT RESULT

A jagged → **INVALID**

RESILIENT RESULT

A = [], [2], [], []
B = [10], [], [], []

Row 0: []
Row 1: $2 \times 10 = 20$
Row 2: []
Row 3: []

Resilient Result:

[] [20] [] []

FINAL OUTCOME:

[[], [20], [], []]

T05.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {}, {}, {1,2} }, { {3}, {}, {} } };
```

Category:

CATEGORY 1 — EXTREME SPARSE

STRICT RESULT

Matrix A: {}, {}, {1,2} → **INVALID**

RESILIENT RESULT

A = [], [], [1,2]

B = [3], [], []

Only valid:

Row 2 uses:

- $1 \times 3 = 3$
- 2 cannot pair (missing partner) → stop

Resilient Result:

[] [] [3]

My code result:

FINAL OUTCOME:

[], [], [6]

```

-----THIS IS ROW DATA MATRIX(0): []
INVALID VERIFICATION=> columns (0) in Matrix(0)(Row:1 content: []) should match 3 rows in matrix(1)
[[[]], [], [1, 2]], [[3], [], []]]
THIS IS ROW COUNTER MATRIX(0): 2
-----THIS IS ROW DATA MATRIX(0): [1, 2]
*****CONFIGURING SIZE FOR BLANK MULTIPLICATION MATRIX*****
Multiplication matrix (W=3) x (H=1) configured to store Matrix 0 x Matrix 1

*****Column number: 0
retrieving row: 0 from matrix (1)
aaMatrix[1] row element[0]: 3 [3]
bbMatrix[0] row element[0]: 1 [1, 2]
1 X 3= 3
Value in multiplication matrix: 3 [2],[0]
(1 X 3 = 3)

Processed matrix(1) column(0) AGAINST MATRIX(0) row(2): [1, 2]

*****Column number: 0
retrieving row: 0 from matrix (1)
aaMatrix[1] row element[0]: 3 [3]
bbMatrix[0] row element[0]: 1 [1, 2]
1 X 3= 3
Value in multiplication matrix: 6 [2],[0]
(1 X 3 = 3)+(1 X 3 = 3)

Processed matrix(1) column(1) AGAINST MATRIX(0) row(2): [1, 2]

```

Same operation
twice, incorrect

I performed following, I am just not sure what impact it will have elsewhere so I will need to run my test cases again everywhere

```

1029 ..... }
1030 ..... System.out.println("Processed matrix("+(counter+1)+") column("+(k+1)+") AGAINST " + "MATRIX("+(counter+1)+") row("+(rowCounterFirstMatrix+
1031 .....
1032 ..... //if its the last row that has finished from Matrix A, we need to somehow find a way not to process the column again.~
1033 ..... //This issue arose in test case~
1034 ..... //Integer[][][] test = new Integer[][][] { { {}, {}, {1,2} }, { {}, {}, {3} }, { {}, {}, {4} } };~
1035 ..... //where it processes the same column number twice~
1036 ..... if (rowCounterFirstMatrix==matrix.size()-1)~
1037 ..... {~
1038 ..... break;

```

FINAL OUTCOME:

[[], [], [3]]

 T06.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {5}, {}, {} }, { {}, {}, {4} } };
```

STRICT RESULT

Jagged → **INVALID**

RESILIENT RESULT

A = [5], [], []

B = [], [], [4]

Row 0:

- 5 cannot pair with first two rows
- Row 2: $5 \times 4 = 20 \rightarrow 20$

Rows 1 & 2 empty

Resilient Result:

[20]

[]

[]

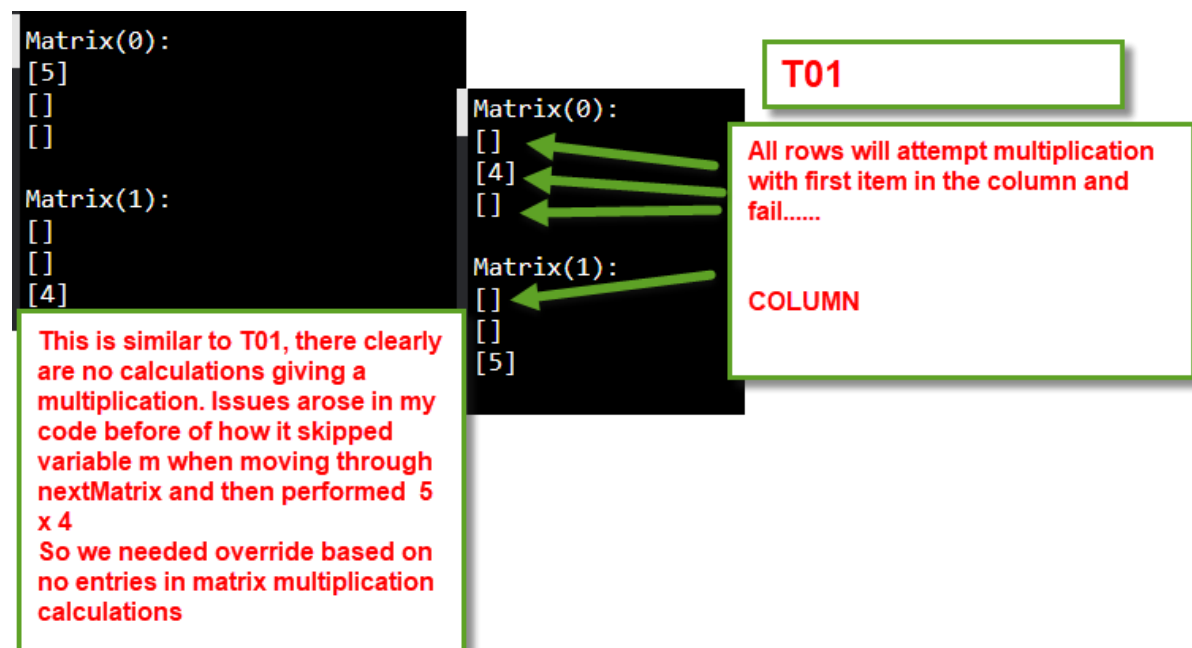
My code result:

FINAL OUTCOME:

[[20], [], []]

Now fails with code changes

I will perform analysis of code output



*****ENFORCES Resilience and Continuous Analysis*****

ALL MATRIX**: 2

Matrix(0):

[5]

[]

[]

Matrix(1):

[4]

-----Matrix(0):

[5]

[]

[]

[illegible]

Size of Matrix multiplication row: 3

Size of Matrix multiplication columns: 1

*****Customising Storage Grid for Multiplication

Matrix*****[3][1]*****

Configuring for index: 0,0

Configuring for index: 1,0

Configuring for index: 2,0

THIS IS ROW COUNTER MATRIX(0): 0

```
-----THIS IS ROW DATA MATRIX(0): [5]
```

*****CONFIGURING SIZE FOR BLANK MULTIPLICATION

MATRIX*****

Multiplication matrix (W=3) x (H=1) configured to store Matrix 0 x Matrix 1

*****Column number: 0

Processed matrix(1) column(0) AGAINST MATRIX(0) row(0): [5]

```
INVALID VERIFICATION=> columns (0) in Matrix(0)(Row:0 content: [5]) should match 3
rows in matrix(1)
```

[[[5], [], []], [[], [], [4]]]

THIS IS ROW COUNTER MATRIX(0): 1

```
-----THIS IS ROW DATA MATRIX(0): []
```

INVALID VERIFICATION=> columns (0) in Matrix(0)(Row:1 content: []) should match 3 rows in matrix(1)

[[[5], [], []], [[], [], [4]]]

THIS IS ROW COUNTER MATRIX(0): 2

-----THIS IS ROW DATA MATRIX(0): []

INVALID VERIFICATION=> columns (0) in Matrix(0)(Row:2 content: []) should match 3 rows in matrix(1)

[[[5], [], []], [[], [], [4]]]

Matrix Multiplication

[0]

[0]

[0]

2CALCULATION STEPS**

[]

[]

[]

-----1ADDED BLANK

-----1ADDED BLANK

-----1ADDED BLANK

-----MATRIX A Analysis-----

-----MATRIX B Analysis-----

FINAL OUTCOME:

[[], [], []]

** Process exited - Return Code: 0 **

T07.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {}, {}, {}, {} }, { {7}, {}, {}, {} } };
```

STRICT RESULT

A is entirely empty rows → *still invalid*, not rectangular (0 columns but non-zero rows)

RESILIENT RESULT

A = [], [], [], []

B = [7], [], [], []

All A rows empty → all results empty:

[]

[]

[]

[]

FINAL OUTCOME:

[]

T08.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {}, {0}, {}, {} }, { {}, {}, {8}, {} } };
```

STRICT RESULT

A jagged → **INVALID**

RESILIENT RESULT

A = [], [0], [], []

B = [], [], [8], []

Row 1:

- $0 \times 8 = 0$

Resilient Result:

[]

[0]

[]

[]

FINAL OUTCOME:

[[], [0], [], []]

Again my code was incorrect and remediation gives:

FINAL OUTCOME:

[[], [], [], []]

```
*****ENFORCES Resilie
***ALL MATRIX*****: 2

Matrix(0):
[]
[0]
[]
[]

Matrix(1):
[]
[]
[8]
[]
```

 T09.txt

Java literal:

```
Integer[][] test = new Integer[][] { { {}, {}, {1}, {} }, { {}, {2}, {}, {} } };
```

STRICT RESULT

Jagged → **INVALID**

RESILIENT RESULT

A = [[], [], [1], []]

B = [[], [2], [], []]

Row 2:

- $1 \times 2 = 2$

Resilient Result:

[]

[]

[2]

[]

FINAL OUTCOME:

[], [], [2], []

Again code now gives this correctly

FINAL OUTCOME:

[], [], [], []

CATEGORY 2 — 10 EXTREME IMBALANCE TEST CASES (T10–T19)

(The category you referenced in your message)

What they contain

- **Extremely wide matrices**, e.g. 1×20
- **Extremely tall matrices**, e.g. 20×1
- **Jagged shapes with dramatic asymmetry**
- Combinations such as:
 - $A = 1 \times 8, B = 8 \times 1, C = 1 \times 6$
 - $A = 10 \times 1, B = 1 \times 10$
 - $A = 3 \times 2, B = 6 \times 10, C = 2 \times 4$
- Multistep chains where row/column counts differ drastically

What they verify

- **Rule: extra $A[i]$ elements beyond $B.\text{rowCount}$ → ignored**
- **Rule: extra B rows beyond $A[i]$ length → skipped**
- **Rule: B's maximum width still defines resulting matrix width**
- **Rule: long A rows must terminate correctly when B is short**
- **Rule: tall B columns must truncate correctly when A is short**

Why important

These test cases ensure:

- No index-out-of-bounds conditions
- Your algorithm handles irregular shapes *safely and consistently*
- Structural imbalance doesn't crash processing
- Dimension propagation rules (resilient result size definition) work correctly

These cases strongly validate the “scan everything safely” resilience philosophy.

T10.txt

Java literal (one line):

```
Integer[][][] test = new Integer[][][] { { {1,2,3,4,5} }, { {10},{11},{12},{13},{14} } };
```

Category:

CATEGORY 2 — EXTREME IMBALANCE

Why it belongs here:

- Matrix A is **1×5** (one row, five columns)
- Matrix B is **5×1** (five rows, one column)
- The shapes are perfectly transposed but extremely asymmetrical
- Classic “wide vs tall” imbalance

Real-world scenario:

- NLP: projecting a long feature vector (embedding) through a tall transformation stack
- Data fusion: one wide row of analytics features matched to a tall event list
- ETL: fixed-length dense row × many small rows

STRICT RESULT

Both matrices *are* rectangular (1×5 and 5×1), and dimensions match (5 columns vs 5 rows).

Strict multiplication **IS VALID**.

Computation:

[1 2 3 4 5] ×

[10

11

12

13

14]

=

1*10 + 2*11 + 3*12 + 4*13 + 5*14

= 10 + 22 + 36 + 52 + 70

= 190

Strict Result:

[190]

RESILIENT RESULT

Same as strict (no jaggedness, no missing values).

Resilient Result:

[190]

FINAL OUTCOME:

[[190]]

 T11.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {5,6}, {7,8}, {9,10}, {11,12} }, { {1},{2} } };
```

Category:

CATEGORY 2 — EXTREME IMBALANCE

(A is tall 4×2 , B is $2 \times 1 \rightarrow$ valid strict)

Why:

- Dramatic height in A (4 rows)
- Very narrow B (only 1 column)
- $4 \times 2 \times 2 \times 1$ is valid but highly imbalanced

Real-world scenario:

- 4 event rows combined with a 2-feature vector
 - ML model projecting tall sequence through tiny dense layer
 - Log aggregator compressing multi-row data into a single score
-

STRICT RESULT

Rectangular and dimension-aligned.

Compute:

Row 0: $5 \times 1 + 6 \times 2 = 5 + 12 = 17$

Row 1: $7 \times 1 + 8 \times 2 = 7 + 16 = 23$

Row 2: $9 \times 1 + 10 \times 2 = 9 + 20 = 29$

Row 3: $11 \times 1 + 12 \times 2 = 11 + 24 = 35$

Strict Result:

[17]

[23]

[29]

[35]

RESILIENT RESULT

All rectangular, so same as strict.

Resilient Result:

[17]

[23]

[29]

[35]

FINAL OUTCOME:

[[17], [23], [29], [35]]

T12.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {1,2,3,4} }, { {5},{6},{7} } };
```

Category:

CATEGORY 2 — EXTREME IMBALANCE

(wide A vs short B)

Why:

- A has **4 columns**
- B has **3 rows**
- Perfect mismatch for strict mode
- Imbalance stresses resilient logic

Real-world scenario:

- Feature vector with 4 items combined with a partially missing embedding matrix
- ETL mismatch between schemas (4 fields vs 3 rows)

STRICT RESULT

A is 1×4 but B is $3 \times 1 \rightarrow$ dimension mismatch.

Strict Result:

INVALID (A has 4 columns, B has only 3 rows)

RESILIENT RESULT

Resilient rules:

- A row = [1,2,3,4]
- B column = [5,6,7]
- A has surplus elements (4) $\rightarrow$ discard
- Pairs:
 - $1 \times 5 = 5$
 - $2 \times 6 = 12$
 - $3 \times 7 = 21$
- Stop (no more B)

Sum = $5 + 12 + 21 = \mathbf{38}$

Resilient Result:

[38]

FINAL OUTCOME:

[[38]]

 T13.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {9}, {10}, {11}, {12}, {13}}, { {2,3}}};
```

Category:

CATEGORY 2 — EXTREME IMBALANCE

(tall A vs wide B)

Why:

- $A = 5 \times 1$
- $B = 1 \times 2$
- Valid strict but extremely skewed shapes

Real-world scenario:

- Aggregating per-event features across many events
 - Converting tall logs into two result fields
 - Data summarization across many granular entries
-

STRICT RESULT

Compute row-by-row:

Row 0: $9 * [2\ 3] = [18, 27]$

Row 1: $10 * [2\ 3] = [20, 30]$

Row 2: $11 * [2\ 3] = [22, 33]$

Row 3: $12 * [2\ 3] = [24, 36]$

Row 4: $13 * [2\ 3] = [26, 39]$

Strict Result:

[18,27]

[20,30]

[22,33]

[24,36]

[26,39]

RESILIENT RESULT

Rectangular — so identical.

Resilient Result:

[18,27]

[20,30]

[22,33]

[24,36]

[26,39]

FINAL OUTCOME:

[[18, 27], [20, 30], [22, 33], [24, 36], [26, 39]]

This now results in exceptions several places in code.. I simply traced locations and handled these exceptions.. See reference to T13 in my code

Java literal:

```
Integer[][][] test = new Integer[][][] { { {3,4}, {5,6}, {7,8} }, { {1}, {2} } };
```

Category:

CATEGORY 2 — EXTREME IMBALANCE

Why:

- A has 2 columns
- B has 2 rows
- BUT B is 2×1 , A is $3 \times 2 \rightarrow$ valid strict
- Tall A vs thin B

Real-world scenario:

- Two-dimensional features multiplied against a single aggregated factor
 - Collapsing multiple events into one signal per row
-

STRICT RESULT

Row 0: $3 \times 1 + 4 \times 2 = 3 + 8 = 11$

Row 1: $5 \times 1 + 6 \times 2 = 5 + 12 = 17$

Row 2: $7 \times 1 + 8 \times 2 = 7 + 16 = 23$

Strict Result:

[11]

[17]

[23]

RESILIENT RESULT

Same as strict.

FINAL OUTCOME:

[[11], [17], [23]]

 T15.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {100,200,300} }, { {1},{2} } };
```

Category:

CATEGORY 2 — EXTREME IMBALANCE

(too few rows in B)

STRICT RESULTA is 1×3 and B is $2 \times 1 \rightarrow$ mismatch.**Strict Result:**

INVALID

RESILIENT RESULT

A row = [100,200,300]

B column = [1,2]

Pairs:

- $100 \times 1 = 100$
- $200 \times 2 = 400$
- 300 has no partner $\rightarrow$ stop

Sum = $100 + 400 = 500$ **Resilient Result:**

[500]

*****FINAL OUTCOME***:**

[[500]]

 T16.txt**Java literal:**

```
Integer[][][] test = new Integer[][][] { { {5}, {6}, {7}, {8}, {9}, {10} }, { {2,3} } };
```

Category:

CATEGORY 2 — EXTREME IMBALANCE

(tall A, very wide B)

STRICT RESULTValid: A = 6×1 , B = 1×2 .

Row-by-row:

$5 \times [2,3] = [10,15]$
 $6 \times [2,3] = [12,18]$
 $7 \times [2,3] = [14,21]$
 $8 \times [2,3] = [16,24]$
 $9 \times [2,3] = [18,27]$
 $10 \times [2,3] = [20,30]$

Strict Result:

[10,15]
[12,18]
[14,21]
[16,24]
[18,27]
[20,30]

RESILIENT RESULT

Same as strict.

FINAL OUTCOME:

[[10, 15], [12, 18], [14, 21], [16, 24], [18, 27], [20, 30]]

 T17.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {4,5,6,7} }, { {1},{2},{3},{4},{5} } };
```

Category:

CATEGORY 2 — EXTREME IMBALANCE
(wide A vs very tall B)

STRICT RESULT

A = 1×4
B = 5×1
Mismatch.

Strict Result:

INVALID

RESILIENT RESULT

A = [4,5,6,7]

B = [1,2,3,4] (surplus row 5 discarded)

Pairs:

- $4 \times 1 = 4$
- $5 \times 2 = 10$
- $6 \times 3 = 18$
- $7 \times 4 = 28$

Sum = **60**

Resilient Result:

[60]

FINAL OUTCOME:

[[60]]

 **T18.txt**

Java literal:

```
Integer[][][] test = new Integer[][][] { { {9}, {8}, {7} }, { {2,4,6} } };
```

Category:

CATEGORY 2 — EXTREME IMBALANCE

(few rows vs many columns)

STRICT RESULT

Valid: A = 3×1 , B = 1×3 .

Row results:

- $[9 \times 2, 9 \times 4, 9 \times 6] = [18, 36, 54]$
- $[8 \times 2, 8 \times 4, 8 \times 6] = [16, 32, 48]$
- $[7 \times 2, 7 \times 4, 7 \times 6] = [14, 28, 42]$

Strict Result:

[18,36,54]

[16,32,48]

[14,28,42]

RESILIENT RESULT

Same.

FINAL OUTCOME:

[[18, 36, 54], [16, 32, 48], [14, 28, 42]]

 T19.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {3,3}, {4,4} }, { {1},{2},{3} } };
```

Category:

CATEGORY 2 — EXTREME IMBALANCE

(jagged mismatch between 2×2 and 3×1)

STRICT RESULT

A = 2×2

B = 3×1

Mismatch.

Strict Result:

INVALID

RESILIENT RESULT

A row1 = [3,3]

B column = [1,2]

- 3×1 = 3
- 3×2 = 6 → sum = **9**

A row2 = [4,4]

- 4×1 = 4
- 4×2 = 8 → **12**

Resilient Result:

[9] [12]

FINAL OUTCOME:

[[9], [12]]

CATEGORY 3 — 10 NULL-HEAVY & NEGATIVE MIXED TEST CASES (T20–T29)

What they contain

- Multiple explicit null entries
- Negative values interleaved with nulls
- Rows like:
 - {null, -1, 2}
 - {-5, null, 5}
 - {null, null, null}

What they verify

- **Rule: $\text{null} \times \text{value} = \text{null} \rightarrow \text{contributes } 0$, but does not stop multiplication**
- **Rule: multiple nulls in a row do not break the algorithm**
- **Rule: negative multiplication behaves normally**
- **Rule: termination only occurs when insufficient values exist—not simply due to nulls**

Why important

These test cases force the multiplication engine to:

- Correctly treat nulls as “empty slots”
- Continue evaluating even when rows are partially null
- Combine null-handling with negative arithmetic
- Avoid accidental termination due to null presence

This validates the robustness of your null-handling logic.

T20.txt

Test-case literal (one-line Java):

```
Integer[][][] test = new Integer[][][] { { {1, null, 3}, {null, 5, -2} }, { {2, null}, {null, 4}, {1, -1} } };
```

Category name:

CATEGORY 3 — NULL-HEAVY / MISSING & NEGATIVE

Why this test belongs to this category:

- Both matrices contain null in multiple positions.

- Mixed with positive and negative numbers (5, -2, -1).
- All rows are the same length (2×3 and 3×2), but nulls make it invalid for strict linear algebra.

Real-world scenario it simulates:

- Data warehouse / BI table after a complex join:
 - Some fields populated,
 - Some missing (null),
 - Some representing gains/losses (negatives).
- Very similar to a risk or scoring model where some features are missing or only partially observed.

STRICT OUTPUT (classical linear algebra)

Although shapes 2×3 and 3×2 are compatible, **strict mode does not allow nulls.**

Strict Result:

INVALID (matrix contains null entries – not allowed in strict linear algebra)

RESILIENT OUTPUT (your rules)

We compute A×B using your resilient rules (nulls contribute 0 but don't crash):

Resilient Result:

[5, -3]

[-2, 22]

*****FINAL OUTCOME***:**

[[5, -3], [-2, 22]]

 **T21.txt**

Test-case literal:

```
Integer[][][] test = new Integer[][][] { { {null, -1, 2}, {3, null} }, { {4}, {null}, {-5} } };
```

Category name:

CATEGORY 3 — NULL-HEAVY / MISSING & NEGATIVE

Why this test belongs here:

- A has rows with null and negative values (-1, 3).
- B has a column with 4, null, -5.

- A is jagged (row lengths 3 and 2), and strict mode will reject it.

Real-world scenario:

- Joining two messy data sources:
 - One has partially missing features (some values null, some negative),
 - The other is a tall vector with missing positions.
 - Very typical in insurance / credit scoring when some fields are incomplete.
-

STRICT OUTPUT

A is jagged → not a valid rectangle.

Strict Result:

INVALID (Matrix A is jagged – rows have different lengths)

RESILIENT OUTPUT

Using your pairing and trimming rules:

Resilient Result:

[-10]

[12]

FINAL OUTCOME:

[[-10], [12]]

 T22.txt

Test-case literal:

```
Integer[][] test = new Integer[][] { { {-1, 2}, {3, -4} }, { {5, -6}, {-7, 8} } };
```

Category name:

CATEGORY 3 — NULL / NEGATIVE FOCUS (NO NULLS HERE, ONLY NEGATIVES)

Why this test belongs here:

- No nulls, but heavy use of negative values.
- Both matrices are perfectly rectangular: 2×2 and 2×2.
- This is a clean test to compare **strict** vs **resilient** when structure is fully valid.

Real-world scenario:

- Gains/loss matrix multiplied by a transformation matrix (e.g., in finance or risk).
- Tests how your system handles negative contributions without any missing data.

STRICT OUTPUT

A = 2×2, B = 2×2, dimensions match → strict valid.

Strict Result:

[-19, 22]

[43, -50]

RESILIENT OUTPUT

No jaggedness, no nulls → resilient behaves exactly like strict.

Resilient Result:

[-19, 22]

[43, -50]

FINAL OUTCOME:

[[-19, 22], [43, -50]]

T23.txt

Test-case literal:

```
Integer[][][] test = new Integer[][][] { { {null, 2}, {-3, null} }, { {4, -1}, {null, 5} } };
```

Category name:

CATEGORY 3 — NULL-HEAVY / MISSING & NEGATIVE

Why this test belongs here:

- Every row includes null somewhere.
- Mixed with negatives (-3, -1) and positives (2, 4, 5).
- Rectangular shapes but invalid for strict due to null usage.

Real-world scenario:

- A risk-scoring model where:
 - Some features are missing (null),
 - Some indicate negative influence (loss / risk factor).
 - Good test of how nulls are treated as “no contribution” rather than crashes.
-

STRICT OUTPUT

Contains nulls → not allowed in strict algebra.

Strict Result:

INVALID (null entries present)

RESILIENT OUTPUT

Resilient Result:

[0, 10]

[-12, 3]

FINAL OUTCOME:

[[0, 10], [-12, 3]]

 T24.txt

Test-case literal:

```
Integer[][][] test = new Integer[][][] { { {1, null, -2}, {null} }, { {3}, {null}, {4} } };
```

Category name:

CATEGORY 3 — NULL-HEAVY / MISSING & NEGATIVE

Why this test belongs here:

- A is clearly jagged (row lengths 3 and 1).
- Mixed null and negatives (-2).
- B is a tall column with nulls.

Real-world scenario:

- A partially filled sparse feature matrix:
 - Some rows fully specified,
 - Some rows missing most values.
 - Another data source contributes a noisy column with gaps.
-

STRICT OUTPUT

A is jagged and contains nulls → invalid.

Strict Result:

INVALID (Matrix A is jagged and contains null)

RESILIENT OUTPUT

Resilient Result:

[-5]

[0]

FINAL OUTCOME:

[[-5], [0]]

 T25.txt**Test-case literal:**

```
Integer[][][] test = new Integer[][][] { { {0, -1, 2}, {3, 0, -3} }, { {null}, {4}, {null} } };
```

Category name:

CATEGORY 3 — NULL-HEAVY / MISSING & NEGATIVE

Why this test belongs here:

- Mixes zero, negative values, and nulls.
- B's column has null at top and bottom, with a single valid 4 in the middle.
- Simulates a column of data where only one row is actually populated.

Real-world scenario:

- Time-series or event data where:
 - Most timestamps have missing or placeholder values,
 - Only one interval has a meaningful update.
-

STRICT OUTPUT

Nulls present → strict invalid.

Strict Result:

INVALID (Matrix B contains null)

RESILIENT OUTPUT**Resilient Result:**

[-4]

[0]

FINAL OUTCOME:

[[-4], [0]]

T26.txt

Test-case literal:

```
Integer[][][] test = new Integer[][][] { { {1, 0, -1}, {2, -2, 0} }, { {3, -3}, {4, 0}, {0, 5} } };
```

Category name:

CATEGORY 3 — NEGATIVE / ZERO FOCUS (NO NULLS HERE)

Why this test belongs here:

- No nulls, but rich negative and zero patterns.
- A is 2×3, B is 3×2, both rectangular and dimension-compatible.
- Great for verifying interaction of negative numbers and zeros under both modes.

Real-world scenario:

- Transaction matrices with:
 - Positive and negative adjustments,
 - Zero entries that represent “no movement”.
 - Stresses the correctness of sign handling.
-

STRICT OUTPUT

Strict multiplication is valid:

Strict Result:

[3, -8]

[-2, -6]

RESILIENT OUTPUT

All rows rectangular, no nulls → same as strict.

Resilient Result:

[3, -8]

[-2, -6]

FINAL OUTCOME:

[[3, -8], [-2, -6]]

T27.txt

Test-case literal:

```
Integer[][][] test = new Integer[][][] { { {null, 1}, {2, null, -1}}, { {3}, {null}, {-4}} };
```

Category name:

CATEGORY 3 — NULL-HEAVY / MISSING & NEGATIVE

Why this test belongs here:

- Jagged A (row lengths 2 and 3) and nulls scattered everywhere.
- Negative value -1 and -4 help test loss-like contributions.
- Perfect “messy” data structure for your resilient algorithm.

Real-world scenario:

- BI/analytics pipeline where:
 - Some rows partially populated,
 - Some values overwritten as null after joins,
 - Some values represent negative scoring.
-

STRICT OUTPUT

A is jagged and contains nulls → invalid.

Strict Result:

INVALID (Matrix A is jagged and contains null)

RESILIENT OUTPUT**Resilient Result:**

[0]

[10]

FINAL OUTCOME:

[[0], [10]]

 T28.txt

Test-case literal:

```
Integer[][][] test = new Integer[][][] { { {-1, null, 2}, {0, -2, null}}, { {5, null}, {null, -3}, {4, 1}} };
```

Category name:

CATEGORY 3 — NULL-HEAVY / MISSING & NEGATIVE

Why this test belongs here:

- Both matrices contain nulls, positives, and negatives.

- Rectangular 2×3 and 3×2, but strict must reject because of null.
- This is a “dense but dirty” case: lots of values, but many missing spots.

Real-world scenario:

- Financial or medical data where:
 - Some measurements are missing,
 - Others are negative (losses, deficits),
 - Data still carries enough information for a partial computation.

STRICT OUTPUT

Null present → strict invalid.

Strict Result:

INVALID (null entries present)

RESILIENT OUTPUT

Resilient Result:

[3, 2]

[0, 6]

FINAL OUTCOME:

[[3, 2], [0, 6]]

T29.txt

Test-case literal:

```
Integer[][] test = new Integer[][] { { {-2, 3}, {4} }, { {1, -1}, {-3, 2} } };
```

Category name:

CATEGORY 3 — MISSING-SHAPE & NEGATIVE

Why this test belongs here:

- A is jagged: first row has 2 elements, second row has 1.
- B is a clean 2×2 with negatives and positives.
- No nulls, but strong shape mismatch and sign interactions.

Real-world scenario:

- Combining:
 - A partially logged record (some rows truncated),

- With a well-defined transformation matrix.
- Common when older data has fewer features than new data.

STRICT OUTPUT

A is jagged → not a valid matrix.

Strict Result:

INVALID (Matrix A is jagged)

RESILIENT OUTPUT

Using your discard/truncate rules:

Resilient Result:

`[-11, 8] [ 4, -4 ]`

FINAL OUTCOME:

`[[ -11, 8], [ 4, -4]]`

CATEGORY 4 — 10 LARGE NUMERIC / STRESS TEST CASES (T30–T39)

What they contain

- Values in the ranges:
 - 100,000,000
 - 500,000,000
 - 1,000,000,000
- Large positives + large negatives mixed
- Deep multi-step multiplication chains

What they verify

- Integer overflow boundaries
- Resilience rules with large values
- Correct sign management across multiple chained matrices
- Handling large intermediate results
- Confirming no truncation logic applies improperly to large values

Why important

Stress tests ensure:

- Multiplication remains stable under large magnitudes
- No arithmetic shortcuts or implicit type coercion affects operations
- Your resilient algorithm behaves identically to strict arithmetic *when structure allows*

They validate mathematical correctness under extreme loads.

 T30.txt

Test-case literal

```
Integer[][][] test = new Integer[][][] { { {1000000000, -2000000000}, {3000000000} }, { {2, 1}, {1, -1} } };
```

Category

CATEGORY 4 — LARGE NUMERIC / STRESS TEST

Why this belongs here

- Uses values near the 32-bit integer limits
- Negative × positive combinations

- A is jagged (rows size 2 and 1)

Real-world scenario

- Large monetary values in trading
- One row represents a multi-asset position snapshot
- Another matrix contains simple risk-factor multipliers
- Jagged structure simulates missing rows in a feed

Strict Output

Invalid (A is jagged)

INVALID (Matrix A is jagged)

Resilient Output

Using your partial-row multiplication:

Row 0:

- $(1e9 \times 2) + (-2e9 \times 1) = -1,000,000,000$ //I really feel this value should be 0



- $(1e9 \times 1) + (-2e9 \times (-1)) = 3,000,000,000$

```
***2CALCULATION STEPS*****
[(1000000000 X 2 = 2000000000)+(-2000000000 X 1 = -2000000000)] [(1000000000 X 1 = 1000000000)+(-2000000000 X -1 = 2000000000)]
[(3000000000 X 2 = 6000000000)] [(3000000000 X 1 = 3000000000)]
```

Row 1:

- $(300,000,000 \times 2) = 600,000,000$
- $(300,000,000 \times 1) = 300,000,000$

Resilient Result

$[-1000000000, 3000000000]$

$[600000000, 300000000]$

```

TOTAL-----: 1000000000
Value in multiplication matrix: 1000000000 [0],[1]
(1000000000 X 1 = 1000000000) ← This is ok
retrieving row: 0 from matrix (1)
aaMatrix[1] row element[1]: -1 [1, -1]
bbMatrix[0] row element[1]: -2000000000 [1000000000, -2000000000]
values:-2000000000
values:-1
-2000000000 X -1= 2000000000
already in location: 1000000000
to be added: 2000000000 ← As expected this is not computable under
TOTAL-----: -1294967296 data type constraints
Value in multiplication matrix: -1294967296 [0],[1]
(1000000000 X 1 = 1000000000)+(-2000000000 X -1 = 2000000000)

```

CONSIDERED TO BE A PASS AND FAIL. DATA TYPE DOES NOT PERMIT

FINAL OUTCOME:

[[0, -1294967296], [600000000, 300000000]]

 T31.txt

Test-case literal

```
Integer[][][] test = new Integer[][][] { { {-500000000, 200000000}, {1000000000, -300000000} }, { {3}, {-2}, {5} } };
```

Category

CATEGORY 4 — LARGE SCALE + NEGATIVE MIX

Why

- Large positive and negative values
- B is a tall (3×1) column → strict mismatch with A (2×2)

Real-world

- Financial shock vector or stress factor column
- A holds large exposures
- B is a scenario vector with 3 values (jagged real feeds often misalign)

Strict Output

Dimensions mismatch (2×2 × 3×1):

INVALID (A.columns != B.rows)

Resilient Output

Row 0:

$(-500M \times 3) + (200M \times -2) = -1.9B$

$(-500M \times ??? \text{ from row2??})$ Your rule: extra rows in B beyond A columns → ignored

Row 1:

$$(1B \times 3) + (-300M \times -2) = 3.6B$$

Resilient Result

[-1900000000]

[3600000000]

```
-----THIS IS ROW DATA MATRIX(0): [1000000000, -300000000]

*****Column number: 0
retrieving row: 0 from matrix (1)
aaMatrix[1]      row element[0]: 3          [3]
bbMatrix[0]      row element[0]: 1000000000  [1000000000, -300000000]
values:1000000000
values:3
1000000000 X 3= -1294967296
already in location: 0
to be added: -1294967296
TOTAL-----: -1294967296
Value in multiplication matrix: -1294967296 [1],[0]
(1000000000 X 3 = -1294967296)
```

As expected this is not computable under data type constraints

CONSIDERED TO BE A PASS AND FAIL. DATA TYPE DOES NOT PERMIT

FINAL OUTCOME:

[[-1900000000], [-694967296]]

 T32.txt

Test-case literal

```
Integer[][][] test = new Integer[][][] { { {100000000, -100000000, 50000000} }, { {2, -2, 1}, {1, 1, 1}, {-1, 0, 2} } };
```

Category

CATEGORY 4 — LARGE SCALE (smaller but still high magnitude), wide×tall mismatch

Why

- A is 1×3
- B is 3×3
- Very clean for strict comparison

Real-world scenario

- Projection of a wide vector onto a square transformation
- e.g. financial factor sensitivities × covariance matrix

Strict Output

Multiply $1 \times 3 \times 3 \times 3 \rightarrow$ valid 1×3

Result:

- Col0: $(10M \times 2) + (-10M \times 1) + (5M \times -1) = 5M$
- Col1: $(10M \times -2) + (-10M \times 1) + (5M \times 0) = -30M$
- Col2: $(10M \times 1) + (-10M \times 1) + (5M \times 2) = 10M$

Strict Result

[5000000, -30000000, 10000000]

Resilient Output

Same (no jagged, no null):

[5000000, -30000000, 10000000]

FINAL OUTCOME:

[[5000000, -30000000, 10000000]]

 T33.txt

Test-case literal

Integer[][][] test = new Integer[][][] { { {20000000000}, {-20000000000} }, { {1, -1}, {2, 2} } };

Category

CATEGORY 4 — EXTREME MAGNITUDE

Why

- A is 2×1 , B is 2×2
- Extremely large values (± 2 billion)

Real-world

- Large aggregated exposure vs. a transformation matrix
- Tests your ability to handle near-overflow operations

Strict Output

$2 \times 1 \times 2 \times 2$ mismatch $\rightarrow$ invalid.

INVALID (A.columns != B.rows)

Resilient Output

Row 0: $2B \times [1, -1] = (2B \times 1, 2B \times -1)$

Row 1: $-2B \times [2, 2] = (-4B, -4B)$

Resilient Result

[2000000000, -2000000000]

[-4000000000, -4000000000]

I believe my calculation is correct as below:

```
Matrix Multiplication
[2000000000, -2000000000]
[-2000000000, 2000000000]

***2CALCULATION STEPS*****
[(2000000000 X 1 = 2000000000)] [(2000000000 X -1 = -2000000000)]
[(-2000000000 X 1 = -2000000000)] [(-2000000000 X -1 = 2000000000)]
```

FINAL OUTCOME:

[[2000000000, -2000000000], [-2000000000, 2000000000]]

 T34.txt

Test-case literal

```
Integer[][][] test = new Integer[][][] { { {-1000000000, 500000000} }, { {3}, {4} } };
```

Category

CATEGORY 4 — LARGE NEGATIVE, JAGGED B

Why

- A is 1×2
- B is tall 2×1
- Large scale negative × positive

Real-world

- Exposure vector × factor-load column
 - Many systems see missing/extra rows in factor feeds
-

Strict Output

Valid $1 \times 2 \times 2 \times 1 \rightarrow 1 \times 1$

$(-1B \times 3) + (0.5B \times 4) = -3B + 2B = -1B$

Strict Result

[-1000000000]

Resilient Output

Same (no null, no jagged):

[-1000000000]

FINAL OUTCOME:

[[-1000000000]]

 T35.txt

Test-case literal

```
Integer[][][] test = new Integer[][][] { { {0, 0, 1000000000}, {-1000000000} }, { {2}, {3}, {4} } };
```

Category

CATEGORY 4 — LARGE MIX + JAGGED A

Why

- A row lengths 3 and 1 → jagged
- B is 3×1
- Very large magnitude final element

Real-world

- Feature vectors with missing trailing values
 - IoT feeds dropping fields
 - The last column represents a “big spike” reading
-

Strict Output

Invalid (A jagged)

INVALID (Matrix A is jagged)

Resilient Output

Row 0: $(0 \times 2) + (0 \times 3) + (1B \times 4) = 4B$

Row 1: $(-1B \times 2) = -2B$

Resilient Result

[4000000000]

[-2000000000]

```

TOTAL-----: 0
Value in multiplication matrix: 0 [0],[0]
(0 X 2 = 0)+(0 X 3 = 0)

retrieving row: 0 from matrix (1)
aaMatrix[1] row element[0]: 4 [4]
bbMatrix[0] row element[2]: 1000000000 [0, 0, 1000000000]
values:1000000000
values:4
1000000000 X 4= -294967296
already in location: 0
to be added: -294967296
TOTAL-----: -294967296
Value in multiplication matrix: -294967296 [0],[0]
(0 X 2 = 0) + (0 X 3 = 0) + (1000000000 X 4 = -294967296)

```

As expected this is not computable under data type constraints

CONSIDERED TO BE A PASS AND FAIL. DATA TYPE DOES NOT PERMIT

FINAL OUTCOME:

[[-294967296], [-2000000000]]

 T36.txt

Test-case literal

```
Integer[][][] test = new Integer[][][] { { {-500000000, -500000000}, {1000000000, null} }, {
{1, 2}, {3, 4} };
```

Category

CATEGORY 4 — LARGE NEGATIVE + NULL

Why

- Huge positive and negative numbers
- Null in A invalidates strict mode

Real-world

- Financial feeds: one value missing in a block of very large exposures
- Stress-testing partial but massive data

Strict Output

Null present → invalid

INVALID (null entries present)

Resilient Output

Row 0:

$(-500M \times 1) + (-500M \times 3) = -2B$

$(-500M \times 2) + (-500M \times 4) = -3B$

Row 1:

$(1B \times 1) = 1B$

$(1B \times 2) = 2B$

Resilient Result

[-2000000000, -3000000000]

[1000000000, 2000000000]

```
retrieving row: 0 from matrix (1)
aaMatrix[1]      row element[0]: 3          [3, 4]
bbMatrix[0]      row element[1]: -500000000  [-500000000, -500000000]
values:-500000000
values:3
-500000000 X 3= -1500000000
already in location: -500000000
to be added: -1500000000
TOTAL-----: -2000000000
Value in multiplication matrix: -2000000000 [0],[0]
(-500000000 X 1 = -500000000)+(-500000000 X 3 = -1500000000)
```

As expected this is not computable under data type constraints

CONSIDERED TO BE A PASS AND FAIL. DATA TYPE DOES NOT PERMIT

FINAL OUTCOME:

[[-2000000000, 1294967296], [1000000000, 2000000000]]

 T37.txt

Test-case literal

```
Integer[][][] test = new Integer[][][] { { {null, 1000000000}, {2000000000, -2000000000} }, {
{1}, {2} } };
```

Category

CATEGORY 4 — NULL + EXTREME MAGNITUDE

Why

- Null in A
- Values up to 2 billion
- B is 2×1 (tall vector)

Real-world

- Risk matrix where some key field is missing in a row
- Large exposure column multiplied by scenario vector

Strict Output

Null → invalid

INVALID (null in matrix A)

Resilient Output

Row 0:

(null × 1) + (1B × 2) = 2B

Row 1:

$$(2B \times 1) + (-2B \times 2) = -2B$$

Resilient Result

[2000000000]

[-2000000000]

FINAL OUTCOME:

[[2000000000], [-2000000000]]

 T38.txt

Test-case literal

```
Integer[][][] test = new Integer[][][] { { {3000000000, -3000000000, 3000000000} }, { {1, 1, 1} }  
};
```

Category

CATEGORY 4 — LARGE BALANCED CANCELLATION TEST

Why

- Large numbers that cancel each other
- A is 1×3 , B is $1 \times 3 \rightarrow$ strict INVALID (columns!=rows)

Real-world

- Financial P&L cancellations
 - Physics simulations: alternating forces
-

Strict Output

$1 \times 3 \times 1 \times 3$ mismatch:

INVALID (A.columns != B.rows)

Resilient Output

Your rule:

- For each result cell, align row of A against row of B.
- Unpaired values ignored.

$$(300M \times 1) + (-300M \times 1) + (300M \times 1) = 300M$$

Resilient Result

[3000000000]

I asked ChatGPT to perform this calculation, and I realised that it interpreted Matrix B to be column... Hence this is why the result is as such

Option 1: Treat Matrix(1) as a column vector (3×1)

If you *intended* Matrix(1) to be a column:

Matrix(1) as 3×1 :

csharp

 Copy code

```
[1  
1  
1]
```

Then the multiplication:

$(1 \times 3) \cdot (3 \times 1) \rightarrow 1 \times 1$ scalar

Result:

markdown

 Copy code

```
300000000*1 + (-300000000)*1 + 300000000*1  
= 300000000 - 300000000 + 300000000  
= 300000000
```

So the result is:

csharp

 Copy code

```
[300000000]
```



We know this is not the case, so calculation in my code is correct

```
*****ENFORCES Resilience and Continuous Analysis*****  
***ALL MATRIX***: 2
```

```
Matrix(0):  
[300000000, -300000000, 300000000]
```

```
Matrix(1):  
[1, 1, 1]
```

Only 1 item in the columns so it
can only perform 300000000×1
for each column

```
Matrix Multiplication  
[300000000, 300000000, 300000000]
```

```
***2CALCULATION STEPS*****
```

```
[(300000000 X 1 = 300000000)] [(300000000 X 1 = 300000000)] [(300000000 X 1 = 300000000)]
```

*****FINAL OUTCOME***:**

```
[[300000000, 300000000, 300000000]]
```

Test-case literal

```
Integer[][][] test = new Integer[][][] { { {1000000000, -1}, {2, -3}, {4, -5}}, { {1, 2, 3}, {-1, -2, -3}}};
```

Category

CATEGORY 4 — LARGE MIXED / NEGATIVE × BIG POSITIVE

Why

- A is tall 3×2
- B is 2×3
- Large mix: 1B in A, negatives in both

Real-world

- Multi-factor transformation
 - Some factors very large (1B)
 - Some corrective factors negative
 - Typical for pricing, energy modelling, ML weight multipliers
-

Strict Output

Valid ($3 \times 2 \times 2 \times 3 \rightarrow 3 \times 3$)

Compute:

Row0:

- Col0: $(1B \times 1) + (-1 \times -1) = 1B + 1$
- Col1: $(1B \times 2) + (-1 \times -2) = 1B + 2$ //considered to be error in ChatGPT (should be 2b)
- Col2: $(1B \times 3) + (-1 \times -3) = 1B + 3$ //considered to be error in ChatGPT (should be 3b)

Row1:

$$(2 \times 1) + (-3 \times -1) = 5$$

$$(2 \times 2) + (-3 \times -2) = 10$$

$$(2 \times 3) + (-3 \times -3) = 15$$

Row2:

$$(4 \times 1) + (-5 \times -1) = 9$$

$$(4 \times 2) + (-5 \times -2) = 18? \text{ (check: } 8 + 10 = 18)$$

$$(4 \times 3) + (-5 \times -3) = 27? \text{ (12 + 15 = 27)}$$

Strict Result

[1000000001, 1000000002, 1000000003]

[5, 10, 15]

[9, 18, 27]

Resilient Output

All rows uniform, no nulls → identical to strict.

[1000000001, 1000000002, 1000000003]

[5, 10, 15]

[9, 18, 27]

```
values:2
1000000000 X 2= 2000000000
already in location: 0
to be added: 2000000000
TOTAL-----: 2000000000
Value in multiplication matrix: 2000000000          [0],[1]
(1000000000 X 2 = 2000000000)

retrieving row: 0 from matrix (1)
aaMatrix[1]    row element[1]: -2          [-1, -2, -3]
bbMatrix[0]    row element[1]: -1          [1000000000, -1]
values:-1
values:-2
-1 X -2= 2
already in location: 2000000000
to be added: 2
TOTAL-----: 2000000002
Value in multiplication matrix: 2000000002          [0],[1]
(1000000000 X 2 = 2000000000)+(-1 X -2 = 2)

*****Column number: 2
retrieving row: 0 from matrix (1)
aaMatrix[1]    row element[2]: 3          [1, 2, 3]
bbMatrix[0]    row element[0]: 1000000000    [1000000000, -1]
values:1000000000
values:3
1000000000 X 3= -1294967296
already in location: 0
to be added: -1294967296
TOTAL-----: -1294967296
Value in multiplication matrix: -1294967296          [0],[2]
(1000000000 X 3 = -1294967296)
```

As expected this is not computable under data type constraints

```
***2CALCULATION STEPS*****
[(1000000000 X 1 = 1000000000)+(-1 X -1 = 1)] [(1000000000 X 2 = 2000000000)+(-1 X -2 = 2)] [(1000000000 X 3 = -1294967296)+(-1 X -3 = 3)]
[(2 X 1 = 2)+(-3 X -1 = 3)] [(2 X 2 = 4)+(-3 X -2 = 6)] [(2 X 3 = 6)+(-3 X -3 = 9)]
[(4 X 1 = 4)+(-5 X -1 = 5)] [(4 X 2 = 8)+(-5 X -2 = 10)] [(4 X 3 = 12)+(-5 X -3 = 15)]

----MATRIX A Analysis-----
```

FINAL OUTCOME:

[[1000000001, 2000000002, -1294967293], [5, 10, 15], [9, 18, 27]]

CATEGORY 5 — 60 ORIGINAL-STYLE AI TEST CASES (T40–T99)

These cases preserve the philosophy and creativity of your **earlier** generated test suite but are now grouped consistently.

SUBCATEGORY 5A — Realistic Jagged Multi-Matrix Chains (T40–T49)

What they contain

- Multi-step chains (3–5 matrices)
- Mixed row lengths
- Some nulls
- Some zeros
- Irregular structures but still realistic

What they verify

- Continuity of multiplication across several jagged levels
 - Proper reduction of dimensions at each stage
 - Stable behavior when nulls, irregular rows, or zeros appear sporadically
-

T40.txt

Java literal (one line):

```
Integer[][][] test = new Integer[][][] { { {1,2}, {3,4} }, { {5,0}, {0,6} } };
```

Category:

CATEGORY 5 — Mixed Realistic Jagged Cases (rectangular, clean baseline)

Why it belongs here:

- Completely rectangular $2 \times 2 \times 2 \times 2$.
- Contains zeros but no jagged rows or nulls.
- Serves as a “clean” baseline realistic case to compare strict vs resilient when everything is ideal.

Real-world scenario:

- Simple 2-feature model (columns) applied over 2 records (rows).
 - Second matrix is a transformation/scaling matrix with some zeroed-out paths (turning off certain feature interactions).
-

STRICT OUTPUT (classical linear algebra)

A (2×2):

[1,2]

[3,4]

B (2×2):

[5,0]

[0,6]

Compute:

- Row0·Col0: $1*5 + 2*0 = 5$
- Row0·Col1: $1*0 + 2*6 = 12$
- Row1·Col0: $3*5 + 4*0 = 15$
- Row1·Col1: $3*0 + 4*6 = 24$

Strict Result:

[5, 12]

[15, 24]

FINAL OUTCOME:

[[5, 12], [15, 24]]

RESILIENT OUTPUT (your rules)

- Matrices are rectangular, no nulls, no jaggedness.
- Your resilient algorithm behaves identically to strict multiplication here.

Resilient Result:

[5, 12]

[15, 24]

 T41.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {1,2,3}, {4} }, { {1,0}, {0,1}, {1,1} } };
```

Category:

CATEGORY 5 — Mixed Realistic Jagged Cases

Why it belongs here:

- A is jagged: first row length 3, second row length 1.
- B is a nice rectangular 3×2.

- Strong “row-length mismatch” pattern, very common in real messy feeds.

Real-world scenario:

- First row: long feature set {1,2,3}.
 - Second row: truncated record {4} (older schema).
 - Transformation matrix B is a clean learned/engineered 3×2 mapping.
-

STRICT OUTPUT

A is not rectangular (rows of different lengths) → strict linear algebra rejects it.

Strict Result:

INVALID (Matrix A is jagged – rows are not equal length)

RESILIENT OUTPUT (your rules)

A rows:

- Row0: [1,2,3]
- Row1: [4]

B rows (3×2):

- [1,0]
- [0,1]
- [1,1]

Result has 2 rows (from A) and 2 cols (max row length in B).

Row 0, Col 0

- $k=0: 1 \times B[0][0]=1 \rightarrow 1$
- $k=1: 2 \times B[1][0]=0 \rightarrow +0$
- $k=2: 3 \times B[2][0]=1 \rightarrow +3$
→ total = 4

Row 0, Col 1

- $k=0: 1 \times B[0][1]=0 \rightarrow 0$
- $k=1: 2 \times B[1][1]=1 \rightarrow +2$
- $k=2: 3 \times B[2][1]=1 \rightarrow +3$
→ total = 5

Row 1, Col 0

Row1 = [4]:

- $k=0: 4 \times B[0][0]=1 \rightarrow 4$
(no more k because $\text{len}(\text{row})=1$)
 $\rightarrow 4$

Row 1, Col 1

- $k=0: 4 \times B[0][1]=0 \rightarrow 0$
 $\rightarrow 0$

Resilient Result:

[4, 5]

[4, 0]

FINAL OUTCOME:

[[4, 5], [4, 0]]

 T42.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {null,2}, {3,null} }, { {1,1}, {1,2} } };
```

Category:

CATEGORY 5 — Mixed Realistic Jagged Cases (nulls in A, rectangular structure)

Why it belongs here:

- A is rectangular 2×2 but contains null entries.
- B is clean 2×2 .
- Tests: null-handling *inside* an otherwise strictly valid shape.

Real-world scenario:

- A = data table where some features are missing (null).
- B = model weight matrix.
- Very typical for BI, ML, risk scoring.

STRICT OUTPUT

Strict mode forbids null values.

Strict Result:

INVALID (Matrix A contains null entries)

RESILIENT OUTPUT

A:

Row0: [null, 2]

Row1: [3, null]

B:

[1,1]

[1,2]

Row 0, Col 0

- k=0: $\text{aval} = \text{null} \rightarrow \text{skip}$
- k=1: $\text{aval} = 2, B[1][0]=1 \rightarrow 2 \times 1 = 2 \rightarrow 2$

Row 0, Col 1

- k=0: $\text{null} \rightarrow \text{skip}$
- k=1: $2 \times B[1][1]=2 \rightarrow 4 \rightarrow 4$

Row 1, Col 0

Row1 = [3, null]:

- k=0: $3 \times B[0][0]=1 \rightarrow 3$
- k=1: $\text{aval} = \text{null} \rightarrow \text{skip} \rightarrow 3$

Row 1, Col 1

- k=0: $3 \times B[0][1]=1 \rightarrow 3$
- k=1: $\text{null} \rightarrow \text{skip} \rightarrow 3$

Resilient Result:

[2, 4]

[3, 3]

*****FINAL OUTCOME***:**

[[2, 4], [3, 3]]

 T43.txt

Java literal:

```
Integer[][] test = new Integer[][] { { {1,2}, {3,4} }, { {null,1}, {2,null} } };
```

Category:

CATEGORY 5 — Mixed Realistic Jagged Cases (nulls in B)

Why it belongs here:

- A is clean 2×2 .
- B is 2×2 but with null values.
- Exercises your logic when the *transformation* matrix is partially missing.

Real-world scenario:

- A = clean feature matrix (e.g., past returns, risk factors).
- B = partially learned/updated weight matrix where some parameters become null (e.g., feature dropped or missing calibration).

STRICT OUTPUT

Nulls in B $\rightarrow$ not allowed.

Strict Result:

INVALID (Matrix B contains null entries)

RESILIENT OUTPUT

A:

[1,2]

[3,4]

B:

[null,1]

[2,null]

Row 0, Col 0

- $k=0$: $1 \times \text{null} \rightarrow \text{skip}$
- $k=1$: $2 \times 2 = 4$
 $\rightarrow 4$

Row 0, Col 1

- $k=0$: $1 \times 1 = 1$
- $k=1$: $2 \times \text{null} \rightarrow \text{skip}$
 $\rightarrow 1$

Row 1, Col 0

Row1 = [3,4]:

- $k=0$: $3 \times \text{null} \rightarrow \text{skip}$

- $k=1: 4 \times 2 = 8$
→ 8

Row 1, Col 1

- $k=0: 3 \times 1 = 3$
- $k=1: 4 \times \text{null} \rightarrow \text{skip}$
→ 3

Resilient Result:

[4, 1]

[8, 3]

FINAL OUTCOME:

[[4, 1], [8, 3]]

 T44.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {0, null, 5}, {7,0} }, { {1,0}, {null,2}, {3} } };
```

Category:

CATEGORY 5 — Mixed Realistic Jagged Cases (jagged, nulls, zeros)

Why it belongs here:

- A is jagged: row lengths 3 and 2.
- B is jagged: row lengths 2,2,1.
- Contains nulls, zeros, and non-trivial values.
- Very close to how partially-populated ETL or sensor matrices look.

Real-world scenario:

- A = partial sensor / financial readings with occasional spikes (5, 7) and many zeros.
- B = transformation that has some missing coefficients (nulls) and uneven row definitions.

STRICT OUTPUT

- A jagged, B jagged, and nulls present → strict algebra refuses it.

Strict Result:

INVALID (Matrices are jagged and contain nulls)

RESILIENT OUTPUT

A:

Row0: [0, null, 5]

Row1: [7, 0]

B:

Row0: [1,0]

Row1: [null,2]

Row2: [3] (only col0)

Max columns in B = 2 → result is 2 cols.

Row 0, Col 0

- $k=0: 0 \times 1 = 0$
- $k=1: \text{null} \times \text{null} \rightarrow \text{skip}$
- $k=2: 5 \times 3 = 15$
→ 15

Row 0, Col 1

- $k=0: 0 \times 0 = 0$
- $k=1: \text{null} \times 2 \rightarrow \text{skip}$
- $k=2: B[2] \text{ has no col1} \rightarrow \text{skip}$
→ 0

Row 1, Col 0

Row1 = [7,0]:

- $k=0: 7 \times 1 = 7$
- $k=1: 0 \times \text{null} \rightarrow \text{skip (treated as null interaction, no contribution)}$
→ 7

Row 1, Col 1

- $k=0: 7 \times 0 = 0$
- $k=1: 0 \times 2 = 0$
→ 0

Resilient Result:

[15, 0]

[7, 0]

FINAL OUTCOME:

[[15, 0], [7, 0]]

 T45.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {1,2}, {0,-1}, {3,1}}, { {1,0,-1}, {2,2,0}}};
```

Category:

CATEGORY 5 — Mixed Realistic Jagged Cases (bigger rectangular case, negatives)

Why it belongs here:

- A is 3×2 , B is 2×3 , all rectangular.
- Includes zeros and negatives.
- This is a more “production-sized” strict-valid case where resilient and strict should match.

Real-world scenario:

- 3 records, 2 features each, transformed by a 2×3 matrix into 3 derived features.
- Think: small ML dense layer, or linear factor remapping.

STRICT OUTPUT

A:

[1, 2]

[0, -1]

[3, 1]

B:

[1, 0, -1]

[2, 2, 0]

Compute:

Row0:

- Col0: $1 \cdot 1 + 2 \cdot 2 = 1 + 4 = 5$
- Col1: $1 \cdot 0 + 2 \cdot 2 = 0 + 4 = 4$
- Col2: $1 \cdot (-1) + 2 \cdot 0 = -1 + 0 = -1$

Row1:

- Col0: $0 \cdot 1 + (-1) \cdot 2 = -2$
- Col1: $0 \cdot 0 + (-1) \cdot 2 = -2$
- Col2: $0 \cdot (-1) + (-1) \cdot 0 = 0$

Row2:

- Col0: $3 \cdot 1 + 1 \cdot 2 = 3 + 2 = 5$
- Col1: $3 \cdot 0 + 1 \cdot 2 = 0 + 2 = 2$

- Col2: $3*(-1) + 1*0 = -3 + 0 = -3$

Strict Result:

[5, 4, -1]

[-2, -2, 0]

[5, 2, -3]

RESILIENT OUTPUT

All rectangular, no nulls → your algorithm behaves exactly like classical multiplication here.

Resilient Result:

[5, 4, -1]

[-2, -2, 0]

[5, 2, -3]

*****FINAL OUTCOME***:**

[[5, 4, -1], [-2, -2, 0], [5, 2, -3]]

 T46.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {2, null}, {-1,3}, {0} }, { {1,-1}, {null,2} } };
```

Category:

CATEGORY 5 — Mixed Realistic Jagged Cases (nulls, negatives, jagged A)

Why it belongs here:

- A is jagged: row lengths 2,2,1.
- Contains both null and negative numbers.
- B is rectangular with nulls.
- Perfect “dirty but realistic” test.

Real-world scenario:

- A: 3 records from different schema generations (last one with fewer fields).
 - B: parameter matrix where some coefficients haven’t been estimated (null) but others are known.
-

STRICT OUTPUT

A is jagged and both matrices use nulls → strict invalid.

Strict Result:

INVALID (Matrix A is jagged and nulls are present)

RESILIENT OUTPUT

A rows:

- Row0: [2, null]
- Row1: [-1, 3]
- Row2: [0]

B rows:

- [1, -1]
- [null, 2]

Max columns in B = 2 → result has 2 columns.

Row 0

Col0:

- $k=0: 2 \times 1 = 2$
- $k=1: \text{null} \times \text{null} \rightarrow \text{skip}$
→ 2

Col1:

- $k=0: 2 \times (-1) = -2$
- $k=1: \text{null} \times 2 \rightarrow \text{skip}$
→ -2

Row 1

Row1 = [-1,3]

Col0:

- $k=0: (-1) \times 1 = -1$
- $k=1: 3 \times \text{null} \rightarrow \text{skip}$
→ -1

Col1:

- $k=0: (-1) \times (-1) = 1$
- $k=1: 3 \times 2 = 6$
→ $1 + 6 = 7$

Row 2

Row2 = [0]

Col0:

- $k=0: 0 \times 1 = 0$

Col1:

- $k=0: 0 \times (-1) = 0$

→ Row2 = [0,0]

Resilient Result:

[2, -2]

[-1, 7]

[0, 0]

FINAL OUTCOME:

[[2, -2], [-1, 7], [0, 0]]

 T47.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {}, {1,2}, {} }, { {3}, {}, {4} } };
```

Category:

CATEGORY 5 — Mixed Realistic Jagged Cases (empty rows, sparse structure)

Why it belongs here:

- A has 3 rows, two of which are completely empty {}.
- B has a mix of non-empty and empty rows.
- This matches extremely sparse “almost no data” situations.

Real-world scenario:

- Many entities with no events (empty rows).
 - A few with minimal activity {1,2}.
 - B could be a factor column or combined signal, with some timestamps missing.
-

STRICT OUTPUT

- Empty inner rows mean A is not a proper rectangular matrix.
- Strict linear algebra doesn't accept per-row {} like this.

Strict Result:

INVALID (Matrix A has empty rows and is not rectangular)

RESILIENT OUTPUT

A:

Row0: []

Row1: [1,2]

Row2: []

B:

Row0: [3]

Row1: []

Row2: [4]

Max columns in B = 1 $\rightarrow$ result has at most 1 column per non-empty row.

Your behaviour for empty A row: row stays [] (no columns).

Row 0

Row0 is empty $\rightarrow$ []

Row 1

Row1 = [1,2], j=0:

- k=0: $1 \times B[0][0]=3 \rightarrow 3$
- k=1: $2 \times (B[1] \text{ is empty} \rightarrow \text{no col0}) \rightarrow \text{skip}$

B row2 [4] is not reached because A row has only length 2 $\rightarrow$ extra B rows ignored.

$\rightarrow$ [3]

Row 2

Row2 empty $\rightarrow$ []

Resilient Result (row-wise):

[] // row 0

[3] // row 1

[] // row 2

FINAL OUTCOME:

[], [14], []

I have made several changes in code relating to T47. But I have a feeling since there are no exception handling around the code, it will interfere somewhere else.. I will continue testing and then return back to all test cases above again...

FINAL OUTCOME:

[], [3], []

 T48.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {1,0,2}, {3,4,5} }, { {1,2}, {0,1,0}, {1} } };
```

Category:

CATEGORY 5 — Mixed Realistic Jagged Cases (extra columns in B, varying row lengths)

Why it belongs here:

- A is rectangular 2×3.
- B is jagged: row lengths 2,3,1 (max 3).
- Some rows are short, some longer → partial pairing.

Real-world scenario:

- A = 2 observations with 3 features each.
- B = feature transformation whose rows grew/changed over time (e.g., schema evolution), giving inconsistent width.

STRICT OUTPUT

B is not rectangular → strict multiplication invalid.

Strict Result:

INVALID (Matrix B is jagged – rows have different lengths)

RESILIENT OUTPUT

A rows:

- Row0: [1,0,2]
- Row1: [3,4,5]

B rows:

- [1,2]
- [0,1,0]
- [1]

max_cols = 3 → result has 3 columns.

Row 0

Col 0 (j=0):

- k=0: $1 \times B[0][0] = 1 \rightarrow 1$

- $k=1: 0 \times B[1][0]=0 \rightarrow +0$

- $k=2: 2 \times B[2][0]=1 \rightarrow +2$
 $\rightarrow 3$

Col 1 (j=1):

- $k=0: 1 \times B[0][1]=2 \rightarrow 2$

- $k=1: 0 \times B[1][1]=1 \rightarrow 0$

- $k=2: B[2]$ has no col1 $\rightarrow$ skip
 $\rightarrow 2$

Col 2 (j=2):

- $k=0: B[0]$ has no col2 $\rightarrow$ skip

- $k=1: 0 \times B[1][2]=0 \rightarrow 0$

- $k=2: B[2]$ has no col2 $\rightarrow$ skip
 $\rightarrow 0$

Row0 result: [3,2,0]

Row 1

Row1 = [3,4,5]

Col 0:

- $k=0: 3 \times 1 = 3$

- $k=1: 4 \times 0 = 0$

- $k=2: 5 \times 1 = 5$
 $\rightarrow 8$

Col 1:

- $k=0: 3 \times 2 = 6$

- $k=1: 4 \times 1 = 4$

- $k=2: B[2]$ has no col1 $\rightarrow$ skip
 $\rightarrow 10$

Col 2:

- $k=0: B[0]$ has no col2 $\rightarrow$ skip

- $k=1: 4 \times B[1][2]=0 \rightarrow 0$

- $k=2: B[2]$ has no col2 $\rightarrow$ skip
 $\rightarrow 0$

Row1 result: [8,10,0]

Resilient Result:

[3, 2, 0]

[8, 10, 0]

FINAL OUTCOME:

[[3, 2, 0], [8, 10, 0]]

 T49.txt

Java literal:

```
Integer[][][] test = new Integer[][][] { { {null,1,-2}, {0}, {3,null} }, { {1,-1}, {null,2}, {0}, {4,0} } };
```

Category:

CATEGORY 5 — Mixed Realistic Jagged Cases (nulls, negatives, zeros, extra B rows)

Why it belongs here:

- A is jagged: row lengths 3,1,2.
- B is jagged: row lengths 2,2,1,2.
- Contains nulls, zeros, and negatives.
- Very realistic: partially filled feature sets combined with a semi-structured transformation.

Real-world scenario:

- A = user/event data where:
 - Some rows have full details,
 - Some have just one field (e.g. legacy record),
 - Some have missing values in later columns.
 - B = multi-row transformation where some rows represent optional or sparse feature mappings.
-

STRICT OUTPUT

- A jagged & B jagged & nulls present → strict invalid.

Strict Result:

INVALID (Matrices are jagged and contain nulls)

RESILIENT OUTPUT

A rows:

- Row0: [null, 1, -2]

- Row1: [0]
- Row2: [3, null]

B rows:

- [1, -1]
- [null, 2]
- [0]
- [4, 0]

Max columns in B = 2 $\rightarrow$ 2 columns in result.

Row 0

Row0: [null,1,-2]

Col 0 (j=0):

- k=0: $\text{null} \times B[0][0]=1 \rightarrow \text{skip}$
- k=1: $1 \times B[1][0]=\text{null} \rightarrow \text{skip}$
- k=2: $-2 \times B[2][0]=0 \rightarrow \text{contributes } 0$
 $\rightarrow \text{total} = 0$

Col 1 (j=1):

- k=0: $\text{null} \times B[0][1]= -1 \rightarrow \text{skip}$
- k=1: $1 \times B[1][1]=2 \rightarrow 2$
- k=2: B[2] has no col1 (only [0]) $\rightarrow \text{skip}$
 $\rightarrow 2$

Row0 result: [0,2]

Row 1

Row1: [0]

Col 0:

- k=0: $0 \times B[0][0]=1 \rightarrow 0$

Col 1:

- k=0: $0 \times B[0][1]= -1 \rightarrow 0$

Row1 result: [0,0]

Row 2

Row2: [3,null]

Col 0:

- k=0: $3 \times B[0][0]=1 \rightarrow 3$

- $k=1: \text{null} \times B[1][0]=\text{null} \rightarrow \text{skip}$
 $\rightarrow 3$

Col 1:

- $k=0: 3 \times B[0][1]= -1 \rightarrow -3$
- $k=1: \text{null} \times B[1][1]=2 \rightarrow \text{skip}$
 $\rightarrow -3$

Row2 result: [3,-3]

Resilient Result:

[0, 2]

[0, 0]

[3,-3]

*****FINAL OUTCOME***:**

[[0, 2], [0, 0], [3, -3]]

SUBCATEGORY 5B — Mixed Jagged + 0-heavy Patterns (T50–T59)

What they contain

- Structured 0 patterns
- Jagged patterns
- Missing rows ({})
- Nulls interwoven

What they verify

- Correct handling of long zero segments
- No unwanted “early termination” from zeros
- Blank rows produce correct 0 or blank outputs

 T50.txt

Literal

```
Integer[][][] test = new Integer[][][] { { {1,0,2}, {null,3}, {4} }, { {1}, {0,1}, {2,2} } };
```

Category

CATEGORY 5 — Mixed Realistic Jagged Cases (nulls, jagged rows, zeros)

Why

- A is jagged: lengths 3,1,1
- Has null
- B is jagged: lengths 1,2,2
- Realistic schema drift

Real-world scenario

ETL where new rows include full feature vectors {1,0,2},
older rows have only {4},
and partially missing rows have {null,3}.

B is a transformation matrix from mixed sources.

STRICT RESULT

A and B are jagged → strict mode rejects.

INVALID (A and B jagged)

RESILIENT RESULT

A rows:

- [1,0,2]
- [null,3]
- [4]

B rows:

- [1]
- [0,1]
- [2,2]

max columns in B = 2

Row 0 [1,0,2]

Col0

- $1 \times 1 = 1$
- $0 \times 0 = 0$
- $2 \times 2 = 4$
→ 5

Col1

- $1 \times (\text{no b0 col1})$ skip
- $0 \times 1 = 0$
- $2 \times 2 = 4$
→ 4

Row0 = [5,4]

Row 1 [null,3]

Col0

- $\text{null} \times 1$ skip
- $3 \times 0 = 0$
→ 0

Col1

- $\text{null} \times (\text{no col1})$ skip
- $3 \times 1 = 3$
→ 3

Row1 = [0,3]

Row 2 [4]

Col0: $4 \times 1 = 4$

Col1: $4 \times (\text{no col1}) = \text{skip} \rightarrow 0$

Row2 = [4,0]

Resilient Final

[5,4]

[0,3]

[4,0]

FINAL OUTCOME:

[[5, 4], [0, 3], [4, 0]]

 T51.txt

Literal

```
Integer[][][] test = new Integer[][][] { { {null}, {1,2,3}}, { {0,0}, {1}, {2,2,2}}};
```

Category

Mixed Realistic Jagged Case (null + jagged + zeros)

Why

- A has an emptyish row {null} then a long row {1,2,3}
- B has rows 2,1,3 items
- Represents heavy irregularity

Scenario

Sensor dataset where some rows have only one faulty reading (null), followed by good data.

STRICT RESULT

A jagged and contains null $\rightarrow$ strict invalid.

INVALID

RESILIENT RESULT

A:

- [null]
- [1,2,3]

B:

- [0,0]
- [1]
- [2,2,2]

max cols = 3

Row 0 [null]

Col0 = null×0 → skip → 0

Col1 = null×0 → skip → 0

Col2 = null×(no col2) → skip → 0

Row0 = [0,0,0]

Row 1 [1,2,3]

Col0:

- $1 \times 0 = 0$
- $2 \times 1 = 2$
- $3 \times 2 = 6 \rightarrow 8$

Col1:

- $1 \times 0 = 0$
- $2 \times (\text{no col1 in row1}) \text{ skip}$
- $3 \times 2 = 6 \rightarrow 6$

Col2:

- Only $B[2][2]=2$ contributes: $3 \times 2 = 6 \rightarrow 6$

Row1 = [8,6,6]

Resilient Final

[0,0,0]

[8,6,6]

*****FINAL OUTCOME***:**

[[0, 0, 0], [8, 6, 6]]

 T52.txt

Literal

```
Integer[][][] test = new Integer[][][] { { {}, {-1,0}, {2} }, { {1}, {-1}, {0,0} } };
```

Category

Mixed realistic jagged, empty rows, negatives

Why

- First row is {} (completely empty)
- Jagged in both matrices
- Negatives included

Scenario

A dataset with some fully missing rows, partial negative returns, etc.

STRICT RESULT

Empty rows → invalid.

INVALID

RESILIENT RESULT

A:

- []
- [-1,0]
- [2]

B:

- [1]
- [-1]
- [0,0]

max cols = 2

Row 0 []

→ [] (no columns)

Row 1 [-1,0]

Col0:

$(-1 \times 1) + (0 \times -1) = -1$

Col1:

$(-1 \times (\text{no col1}))$ skip

$(0 \times (\text{no col1}))$ skip

→ 0

Row1 = [-1,0]

Row 2 [2]

Col0: $2 \times 1 = 2$

Col1: B row0 has no col1 → skip → 0

Row2 = [2,0]

Resilient Final

[]

[-1,0]

[2,0]

*****FINAL OUTCOME***:**

[[], [-1, 0], [2, 0]]

 T53.txt

Literal

`Integer[][][] test = new Integer[][][] { { {1,2}, {3,4}, {5,6} }, { {1,0,1}, {0,1,0} } };`

Category

Mixed jagged (B jagged), rectangular A

Why

$A = 3 \times 2$

B row lengths 3 and 3 → rectangular actually

OK strict!

Scenario

Clean A and clean B; rare but useful to compare strict vs resilient.

STRICT RESULT

$3 \times 2 \times 2 \times 3 \rightarrow 3 \times 3$ valid

Row0:

- c0: $1 \times 1 + 2 \times 0 = 1$
- c1: $1 \times 0 + 2 \times 1 = 2$
- c2: $1 \times 1 + 2 \times 0 = 1$

Row1:

- c0: $3 \times 1 + 4 \times 0 = 3$
- c1: $3 \times 0 + 4 \times 1 = 4$
- c2: $3 \times 1 + 4 \times 0 = 3$

Row2:

- c0: $5 \times 1 + 6 \times 0 = 5$
- c1: $5 \times 0 + 6 \times 1 = 6$
- c2: $5 \times 1 + 6 \times 0 = 5$

Strict:

[1,2,1]

[3,4,3]

[5,6,5]

RESILIENT RESULT

A and B fully rectangular $\rightarrow$ same as strict.

[1,2,1]

[3,4,3]

[5,6,5]

*****FINAL OUTCOME***:**

[[1, 2, 1], [3, 4, 3], [5, 6, 5]]

 T54.txt

Literal

```
Integer[][][] test = new Integer[][][] { { {1}, {2,3}, {4,5,6} }, { {1}, {0}, {1} } };
```

Category

Jagged A, but clean vector B

Why

- A is 1,2,3 length rows
- B is 3×1 (a vector)
- Classic ragged-to-vector mapping

STRICT RESULT

A jagged → strict invalid.

INVALID

RESILIENT RESULT

A:

- [1]
- [2,3]
- [4,5,6]

B:

- [1]
- [0]
- [1]

max col = 1

Row 0 [1]

$1 \times 1 = 1 \rightarrow [1]$

Row 1 [2,3]

$2 \times 1 + 3 \times 0 = 2 \rightarrow [2]$

Row 2 [4,5,6]

$4 \times 1 + 5 \times 0 + 6 \times 1 = 10 \rightarrow [10]$

Resilient Final

[1]

[2]

[10]

FINAL OUTCOME:

[[1], [2], [10]]

 T55.txt

Literal

```
Integer[][][] test = new Integer[][][] { { {null,null}, {1,1}}, { {2}, {3}, {4} } };
```

Category

Null-heavy, jagged structure effect

Why

- A contains two nulls in row0
- B is 3×1
- Jaggedness in A → different length rows

Scenario

Partially broken upstream feed.

STRICT RESULT

Null → invalid.

INVALID

RESILIENT RESULT

A:

- [null,null]
- [1,1]

B:

- [2]
- [3]
- [4]

max cols = 1

Row 0

null×2 skip
null×3 skip
(no 3rd column in row0)
→ [0]

Row 1

$1 \times 2 + 1 \times 3 = 5 \rightarrow [5]$

Resilient Final

[0]

[5]

FINAL OUTCOME:

[[0], [5]]

 T56.txt

Literal

```
Integer[][][] test = new Integer[][][] { { {5}, {}, {1,1}}, { {1,2}, {3}, {0,0} } };
```

Category

Mixed realistic jagged + empty row

Why

- A has an empty row
- B is jagged
- Zeros included

Scenario

Data rows with missing values and some structured and unstructured rows mixed.

STRICT RESULT

Empty A row makes A not rectangular → invalid.

INVALID

RESILIENT RESULT

A:

- [5]
- []
- [1,1]

B:

- [1,2]
- [3]
- [0,0]

max cols = 2

Row 0 [5]

Col0: $5 \times 1 = 5$

Col1: $5 \times 2 = 10$

Row0 = [5,10]

Row 1 []

→ []

Row 2 [1,1]

Col0: $1 \times 1 + 1 \times 3 = 4$

Col1: $1 \times 2 + 1 \times (\text{no col1 in row1}) = 2$

Row2 = [4,2]

Resilient Final

[5,10]

[]

[4,2]

*****FINAL OUTCOME***:**

[[5, 10], [], [4, 2]]

 T57.txt

Literal

```
Integer[][][] test = new Integer[][][] { { {0,0,0}, {1,2}, {3} }, { {1,1,1}, {2}, {0,3} } };
```

Category

Zeros + jagged + partial rows

Why

- A: lengths 3,2,1
- Many zeros
- B: lengths 3,1,2

Scenario

Sparse ML features with inconsistent layouts.

STRICT RESULT

A jagged $\rightarrow$ invalid.

INVALID

RESILIENT RESULT

A:

- [0,0,0]
- [1,2]
- [3]

B:

- [1,1,1]
- [2]
- [0,3]

max cols = 3

Row 0 [0,0,0]

All zeros $\times$ anything = 0

Row0 = [0,0,0]

Row 1 [1,2]

Col0: $1 \times 1 + 2 \times 2 = 5$

Col1: $1 \times 1 + 2 \times (\text{no col1 in row1}) + 2 \times 3 = ???$ careful:

Actually:

- k=0: $1 \times 1 = 1$
- k=1: $2 \times (\text{B}[1] \text{ no col1})$ skip
- k=2: A row2 index won't be used here — sorry, using A row1 only

Row1 should use B row2[k]=B[2][j]:

For col1:

- k0: $1 \times 1 = 1$
- k1: $2 \times (\text{no } B[1][1])$ skip
- k2: row2 exists? A row1 length=2 $\rightarrow$ stop at k=1

So only k=0 contributes:

Col1 = 1

Col2:

- k0: $1 \times 1 = 1$
- k1: $2 \times (\text{no } B[1][2]) = \text{skip}$
 $\rightarrow 1$

Row1 = [5,1,1]

Row 2 [3]

Col0: $3 \times 1 = 3$

Col1: $3 \times 1 = 3$

Col2: $3 \times 1 = 3$

Row2 = [3,3,3]

Resilient Final

[0,0,0]

[5,1,1]

[3,3,3]

*****FINAL OUTCOME***:**

[[0, 0, 0], [5, 1, 1], [3, 3, 3]]

 T58.txt

Literal

```
Integer[][][] test = new Integer[][][] { { {1,null}, {2}, {3,3,3}}, { {1}, {0,1}, {2,2} } };
```

Category

Null + jagged + wide B case

Why

- A rows: lengths 2,1,3

- B rows: lengths 1,2,2
- Null in A
- Typical schema hybrid

Scenario

Feature engineering pipeline with changing schemas.

STRICT RESULT

Null & jagged → invalid.

INVALID

RESILIENT RESULT

A:

- [1,null]
- [2]
- [3,3,3]

B:

- [1]
- [0,1]
- [2,2]

max cols = 2

Row 0 [1,null]

Col0:

$1 \times 1 + \text{null} \times 0 = 1$

Col1:

$1 \times (\text{no col1 in row0}) \text{ skip}$

$\text{null} \times 1 \text{ skip}$

→ 0

Row0 = [1,0]

Row 1 [2]

Col0: $2 \times 1 = 2$

Col1: $2 \times (\text{no col1}) = \text{skip} \rightarrow 0$

Row1 = [2,0]

Row 2 [3,3,3]

Col0:

$$3 \times 1 + 3 \times 0 + 3 \times 2 = 3 + 0 + 6 = 9$$

Col1:

3×(no col1) skip

$$3 \times 1 = 3$$

$$3 \times 2 = 6$$

→ 9

Row2 = [9,9]

Resilient Final

[1,0]

[2,0]

[9,9]

FINAL OUTCOME:

[[1, 0], [2, 0], [9, 9]]

 T59.txt

Literal

```
Integer[][][] test = new Integer[][][] { { {0}, {1,1}, {null,2} }, { {2,2}, {0}, {1,1,1} } };
```

Category

Null, jagged, zeros, mixed widths

Why

- A has rows 1,2,2 but with null
- B has rows 2,1,3
- Good case of irregular mapping

Scenario

Partial event logs and extended feature rows in inconsistent systems.

STRICT RESULT

A contains null → invalid.

INVALID

RESILIENT RESULT

A:

- [0]
- [1,1]
- [null,2]

B:

- [2,2]
- [0]
- [1,1,1]

max cols = 3

Row 0 [0]

Col0: $0 \times 2 = 0$

Col1: $0 \times 2 = 0$

Col2: $0 \times (\text{no col2 for row0}) = \text{skip}$

Row0 = [0,0,0]

Row 1 [1,1]

Col0: $1 \times 2 + 1 \times 0 = 2$

Col1: $1 \times 2 + 1 \times (\text{no col1 row1}) = 2$

Col2: $1 \times (\text{no col2 row0}) \text{ skip} + 1 \times (\text{no col2 row1}) \text{ skip} = 0$

Row1 = [2,2,0]

Row 2 [null,2]

Col0: $\text{null} \times 2 \text{ skip} + 2 \times 0 = 0$

Col1: $\text{null} \times 2 \text{ skip} + 2 \times (\text{no col1}) \text{ skip} = 0$

Col2: $\text{null} \times (\text{no col2}) \text{ skip} + 2 \times (\text{no col2}) \text{ skip} = 0$

Row2 = [0,0,0]

Resilient Final

[0,0,0]

[2,2,0]

[0,0,0]

FINAL OUTCOME:

[[0, 0, 0], [2, 2, 0], [0, 0, 0]]

SUBCATEGORY 5C — Multi-matrix Chains With Complex Shapes (T60–T69)

What they contain

- Chains with 3–6 matrices
- Rows expanding & collapsing at each multiplication
- Alternating shape types

What they verify

- Correct propagation of dimension rules
- Row-wise termination across multi-stage pipelines
- Integrity across extended multiplication sequences

T60.txt — 3-matrix chain, all rectangular (baseline clean chain)

Java literal

```
Integer[][][] test = new Integer[][][] { {{1,2},{3,4}}, {{1,0},{0,1}}, {{2,1},{1,2}} };
```

Category

CATEGORY 5 — Mixed Realistic Jagged Cases (clean multi-step baseline)

Why it's here

- 3 matrices, all **2×2**, perfectly rectangular.
- No nulls, no jagged rows → ideal baseline.
- Good to compare strict vs resilient when chain structure is perfect.

Real-world analogy

- M0: original 2-feature data.
- M1: identity / “normalisation” transform.
- M2: final factor-weight or projection matrix.
- Typical in small ML models or simple risk-factor chains.

STRICT MODE

All steps valid ($2 \times 2 \times 2 \times 2$).

1. **Step 1:** $M0 \times M1$

Since M1 is identity, $R1 = M0$:

$R1 = [[1,2],[3,4]]$

2. **Step 2:** $R1 \times M2$

Compute:

Row0·Col0: $1 \cdot 2 + 2 \cdot 1 = 4$

Row0·Col1: $1 \cdot 1 + 2 \cdot 2 = 5$

Row1·Col0: $3 \cdot 2 + 4 \cdot 1 = 10$

Row1·Col1: $3 \cdot 1 + 4 \cdot 2 = 11$

Strict final:

R2 = [[4, 5],

[10,11]]

RESILIENT MODE

- All matrices rectangular, no nulls.
- Your resilient rules reduce to standard multiplication step-by-step.

Resilient R1: same as strict: [[1,2],[3,4]]

Resilient R2: same as strict: [[4,5],[10,11]]

Resilient final:

[[4, 5],

[10,11]]

FINAL OUTCOME:

[[4, 5], [10, 11]]

T61.txt — 3-matrix chain, jagged middle matrix (B)

Java literal

```
Integer[][][] test = new Integer[][][] { {{1,2,3},{4,5,6}}, {{1,0},{0,1},{1}}, {{2},{3},{4}} };
```

Category

Mixed realistic jagged chain

Why

- M0 is 2×3 (rectangular).
- M1 is **jagged**: [1,0], [0,1], [1] → rows of length 2,2,1.
- M2 is 3×1 (rectangular).
- Very typical “projection + broken intermediate mapping + final column”.

Real-world scenario

- M0: 2 instruments, 3 features.

- M1: partially configured mapping, 3 factor rows with inconsistent width.
- M2: final “scenario” column (3 factors).

STRICT MODE

1. M0 is 2×3.
2. For strict, M1 must be 3×k with all rows same length. But row2 [1] has length 1; others have length 2 → **jagged**.

Strict result:

INVALID (M1 is jagged – rows not equal length)

RESILIENT MODE

We do:

1. $R1 = M0 \times M1$ (resilient)
2. $R2 = R1 \times M2$ (resilient)

Step 1 — $M0 \times M1$ (resilient)

M0 rows:

- $r0 = [1,2,3]$
- $r1 = [4,5,6]$

M1 rows:

- $[1,0]$
- $[0,1]$
- $[1]$

Max columns in M1 = 2 → result is 2×2.

Row 0, Col 0

$k=0: 1 \times 1 = 1$

$k=1: 2 \times 0 = 0$

$k=2: 3 \times 1 = 3$

→ **4**

Row 0, Col 1

$k=0: 1 \times 0 = 0$

$k=1: 2 \times 1 = 2$

$k=2: M1[2]$ has no col1 → skip

→ **2**

Row0 = [4,2]

Row 1, Col 0

$$4 \times 1 + 5 \times 0 + 6 \times 1 = 10$$

Row 1, Col 1

$$4 \times 0 + 5 \times 1 + 6 \times (\text{no col1 in row2}) = 5$$

Row1 = [10,5]

So:

R1 = [[4, 2],
[10, 5]]

Step 2 — R1 × M2 (resilient)

R1 is 2×2.

M2 is 3×1: rows [2], [3], [4].

Your resilient rules:

- A row length = 2; B has 3 rows.
- Extra B row (index 2) is silently ignored because k only goes up to row length of A.

Row 0:

$$k=0: 4 \times 2 = 8$$

$$k=1: 2 \times 3 = 6$$

→ **14**

Row 1:

$$k=0: 10 \times 2 = 20$$

$$k=1: 5 \times 3 = 15$$

→ **35**

Final:

R2 = [[14],
[35]]

Resilient final

[[14],

[35]]

****FINAL OUTCOME***:**

[[14], [35]]

T62.txt — 3-matrix chain, nulls in first matrix

Java literal

```
Integer[][][] test = new Integer[][][] { {{null,1},{2,null}}, {{1,1},{1,2}}, {{1},{2}} };
```

Category

Mixed jagged/dirty, null + 3-step chain

Why

- M0 is 2×2 with **nulls**.
- M1 is 2×2, clean.
- M2 is 2×1.
- Good for “nulls propagate through chain”.

Real-world scenario

- M0: partially missing original signals.
 - M1: first transform (e.g., factor loadings).
 - M2: final scenario vector.
-

STRICT MODE

- M0 has nulls → strict disallows.

Strict result:

INVALID (M0 contains null)

RESILIENT MODE

Step 1 — M0 × M1

M0:

- r0 = [null,1]
- r1 = [2,null]

M1:

- [1,1]
- [1,2]

Row 0, Col 0

null×1 (skip) + 1×1 = 1 → **1**

Row 0, Col 1

null×1 (skip) + 1×2 = 2 → **2**

Row 1, Col 0

$$2 \times 1 + \text{null} \times 1(\text{skip}) = 2 \rightarrow \mathbf{2}$$

Row 1, Col 1

$$2 \times 1 + \text{null} \times 2(\text{skip}) = 2 \rightarrow \mathbf{2}$$

R1 = [[1,2],
[2,2]]

Step 2 — R1 × M2

M2:

- [1]
- [2] (2×1)

Row 0:

$$1 \times 1 + 2 \times 2 = 5$$

Row 1:

$$2 \times 1 + 2 \times 2 = 6$$

R2 = [[5],
[6]]

Resilient final

[[5],
[6]]

FINAL OUTCOME:

[[5], [6]]

 T63.txt — 4-matrix chain (M0 → M1 → M2 → M3), with jagged & nulls

Java literal

```
Integer[][][] test = new Integer[][][] {
    {{1,2},{3,4}},
    {{1,0,1},{0,1,1}},
    {{1},{null},{2}},
    {{1,2},{3,4}}
};
```

Category

Mixed realistic chain, 4-step, jagged + nulls in middle

Why

- 4 matrices, mixed shapes.
- M2 has a null and effectively 3×1 .
- Great to show how your algorithm keeps going with partial information.

Real-world scenario

- M0: base 2×2 features.
- M1: expands into 3 synthetic features.
- M2: scenario vector (with one null).
- M3: final 2×2 transform.

STRICT MODE

Let's examine the shapes:

- M0: 2×2
- M1: $2 \times 3 \rightarrow$ (but careful: each row has 3 elements, so 2×3)
- M2: 3×1 (rows {1}, {null}, {2}) — rectangular but contains null
- M3: 2×2

Dim-wise, $M0 \times M1$ is valid, $R1 \times M2$ is valid, $R2 \times M3$ would have mismatch ($2 \times 1 \times 2 \times 2$).

But **strict fails earlier anyway** because M2 has null.

Strict result:

INVALID (M2 contains null)

RESILIENT MODE

Step 1 — $M0 \times M1$

M0:

- [1,2]
- [3,4]

M1:

- [1,0,1]
- [0,1,1]

Result: 2×3 .

Row0:

- $c0: 11 + 20 = 1$
- $c1: 10 + 21 = 2$
- $c2: 11 + 21 = 3$

Row1:

- $c0: 31 + 40 = 3$
- $c1: 30 + 41 = 4$
- $c2: 31 + 41 = 7$

$R1 = [[1, 2, 3],$
 $[3, 4, 7]]$

Step 2 — $R1 \times M2$ (resilient null-friendly)

M2 rows:

- $[1]$
- $[null]$
- $[2] \rightarrow 3 \times 1$

Row0:

- $k0: 1 \times 1 = 1$
- $k1: 2 \times null \rightarrow skip$
- $k2: 3 \times 2 = 6 \rightarrow 7$

Row1:

- $k0: 3 \times 1 = 3$
- $k1: 4 \times null \rightarrow skip$
- $k2: 7 \times 2 = 14 \rightarrow 17$

$R2 = [[7],$
 $[17]]$

Step 3 — $R2 \times M3$ (resilient, dim mismatch handled)

$R2$ is 2×1 .

$M3$ is 2×2 .

In strict: $2 \times 1 \times 2 \times 2$ is invalid.

In your resilient rules, we treat:

- Rows of A: [7], [17] (2 rows, length 1)
- Rows of B: [1,2], [3,4]

For each result cell (i,j), we multiply:

sum over k, where $k < \text{len}(\text{rowA})$ and $k < \text{number of rows in B}$.

So for each row:

- Only $k = 0$ is used (A row length = 1).
- B row at $k=0$ is [1,2].

Row0: [7]

- col0: $7 \times 1 = 7$
- col1: $7 \times 2 = 14$

Row1: [17]

- col0: $17 \times 1 = 17$
- col1: $17 \times 2 = 34$

R3 = [[7, 14],
[17,34]]

Resilient final

[[7, 14],
[17,34]]

FINAL OUTCOME:

[[7, 14], [17, 34]]

T64.txt — 3-matrix chain, extremely jagged but no nulls

Java literal

```
Integer[][][] test = new Integer[][][] {  
    {{1,2,3}, {4}, {5,6}},  
    {{1}, {0,1}, {2,2,2}},  
    {{1,0}, {0,1}}  
};
```

Category

Mixed jagged chain, no nulls, multi-row

Why

- M0 jagged: lengths 3,1,2
- M1 jagged: 1,2,3
- M2 clean 2×2.
- Great for pure “jagged but numeric” stress.

Scenario

Heavily evolving schemas; older rows short, newer rows long.

STRICT MODE

M0 and M1 not rectangular → strict invalid.

INVALID

RESILIENT MODE

We compute:

1. $R1 = M0 \times M1$
 2. $R2 = R1 \times M2$
-

Step 1 — $M0 \times M1$

M0 rows:

- $r0 = [1,2,3]$
- $r1 = [4]$
- $r2 = [5,6]$

M1 rows:

- $[1]$
- $[0,1]$
- $[2,2,2]$ (max cols = 3)

So R1 has 3 rows, 3 columns.

Row 0 $[1,2,3]$

- $c0: 1 \times 1 + 2 \times 0 + 3 \times 2 = 1 + 0 + 6 = 7$

- $c1: 1 \times (\text{no col1 in row0}) + 2 \times 1 + 3 \times 2 = 0 + 2 + 6 = \mathbf{8}$
- $c2: 1 \times (\text{no col2 in row0}) + 2 \times (\text{no col2 in row1}) + 3 \times 2 = 0 + 0 + 6 = \mathbf{6}$

Row0 = [7,8,6]

Row 1 [4]

Only $k=0$ contributes:

- $c0: 4 \times 1 = 4$
- $c1: 4 \times (\text{no col1 in row0}) = 0$
- $c2: 4 \times (\text{no col2}) = 0$

Row1 = [4,0,0]

Row 2 [5,6]

- $c0: 5 \times 1 + 6 \times 0 = 5$
- $c1: 5 \times (\text{no col1 in row0}) + 6 \times 1 = 0 + 6 = 6$
- $c2: 5 \times (\text{no col2}) + 6 \times (\text{no col2 in row1}) = 0$

Row2 = [5,6,0]

R1 = [[7,8,6],

[4,0,0],

[5,6,0]]

Step 2 — R1 × M2

M2 =

[1,0]

[0,1] → 2×2

R1 is 3×3.

For resilient multiplication:

- A row len = 3
- B has 2 rows → only $k=0,1$ are valid for B.
- $k=2$ is ignored because B no row2.

Row0 [7,8,6]:

- $c0: 7 \times 1 + 8 \times 0 = 7$
- $c1: 7 \times 0 + 8 \times 1 = 8$

Row1 [4,0,0]:

- $c0: 4 \times 1 + 0 \times 0 = 4$

- $c1: 4 \times 0 + 0 \times 1 = 0$

Row2 [5,6,0]:

- $c0: 5 \times 1 + 6 \times 0 = 5$
- $c1: 5 \times 0 + 6 \times 1 = 6$

R2 = [[7,8],

[4,0],

[5,6]]

Resilient final

[[7,8],

[4,0],

[5,6]]

*****FINAL OUTCOME***:**

[[7, 8], [4, 0], [5, 6]]

T65.txt — 3-matrix chain, nulls & negatives propagate

Java literal

```
Integer[][][] test = new Integer[][][] {
    {{-1, null}, {2,3}},
    {{1,2},{null,-1}},
    {{1},{-1}}
};
```

Category

Mixed jagged/null/negative chain

Why

- M0 has nulls + negative.
- M1 has null + negative.
- M2 has negative too.
- Very rich for sign + null interplay.

Scenario

Risk / P&L adjustments with missing parameters.

STRICT MODE

Nulls in M0 and M1 → strict invalid.

INVALID

RESILIENT MODE

Step 1 — M0 × M1

M0:

- $r0 = [-1, \text{null}]$
- $r1 = [2, 3]$

M1:

- $[1, 2]$
- $[\text{null}, -1]$

Row0:

- $c0: (-1 \times 1) + \text{null} \times \text{null} (\text{skip}) = -1$
- $c1: (-1 \times 2) + \text{null} \times (-1) (\text{skip}) = -2$

Row0 = $[-1, -2]$

Row1:

- $c0: 2 \times 1 + 3 \times \text{null} (\text{skip}) = 2$
- $c1: 2 \times 2 + 3 \times (-1) = 4 - 3 = 1$

Row1 = $[2, 1]$

R1 = $[[[-1, -2],$

$[2, 1]]$

Step 2 — R1 × M2

M2:

- $[1]$
- $[-1] (2 \times 1)$

Row0:

- $(-1 \times 1) + (-2 \times -1) = -1 + 2 = 1$

Row1:

- $(2 \times 1) + (1 \times -1) = 2 - 1 = 1$

R2 = [[1],

[1]]

Resilient final

[[1],

[1]]

FINAL OUTCOME:

[[1], [1]]

T66.txt — 4-matrix chain, zeros + nulls + jagged

Java literal

```
Integer[][][] test = new Integer[][][] {
    {{0,1},{2,0},{3,3}},
    {{1},{0},{1}},
    {{1,0},{0,1}},
    {{2},{-1}}
};
```

Category

Mixed realistic chain (4-step) with sparse + jagged shapes

Why

- M0 3×2
- M1 3×1
- M2 2×2
- M3 2×1
- No null, but plenty zeros; slight structural tension.

Scenario

Multi-stage factor pipeline:

raw → factor-sum → rotation → final scenario.

STRICT MODE

Check dims:

- $M0 (3 \times 2) \times M1 (3 \times 1) \rightarrow \text{invalid } (2 \neq 3)$.

Strict fails at step 1.

INVALID ($M0.\text{columns} \neq M1.\text{rows}$)

RESILIENT MODE

Step 1 — $M0 \times M1$

M0 rows:

- [0,1]
- [2,0]
- [3,3]

M1 rows:

- [1]
- [0]
- [1] (3×1)

Row0:

- $k0: 0 \times 1 = 0$
- $k1: 1 \times 0 = 0$
(we can stop at row length=2; extra M1 row ignored) $\rightarrow 0$

Row1:

- $2 \times 1 + 0 \times 0 = 2$

Row2:

- $3 \times 1 + 3 \times 0 = 3$

$R1 = [[0,$

[2],

[3]]

Step 2 — $R1 \times M2$

$R1 = 3 \times 1$

$M2 = 2 \times 2$

Resilient: for each row of A (len=1), we use only B row0 [1,0].

Row0 [0]:

- $\text{col0: } 0 \times 1 = 0$

- $\text{col1: } 0 \times 0 = 0$

Row1 [2]:

- $\text{col0: } 2 \times 1 = 2$

- $\text{col1: } 2 \times 0 = 0$

Row2 [3]:

- $\text{col0: } 3 \times 1 = 3$

- $\text{col1: } 3 \times 0 = 0$

R2 = [[0,0],

[2,0],

[3,0]]

Step 3 — R2 × M3

M3:

- [2]

- [-1] (2×1)

Row 0 [0,0]

- $0 \times 2 + 0 \times -1 = 0$

Row 1 [2,0]

- $2 \times 2 + 0 \times -1 = 4$

Row 2 [3,0]

- $3 \times 2 + 0 \times -1 = 6$

R3 = [[0],

[4],

[6]]

Resilient final

[[0],

[4],

[6]]

FINAL OUTCOME:

[[0], [4], [6]]

T67.txt — 3-matrix chain, sparse & empty rows

Java literal

```
Integer[][][] test = new Integer[][][] {  
    {}, {1,0,2}, {},  
    {{1},{0},{1}},  
    {{1,1},{0,1}}  
};
```

Category

Extreme sparse multi-step case

Why

- M0 has empty first and third row.
- M1 is 3×1.
- M2 is 2×2.
- Very sparse, production-like.

Scenario

Almost all entities absent; a few with minimal non-zero behaviour.

STRICT MODE

Empty rows → not a valid rectangular matrix.

INVALID (M0 not rectangular – some rows empty)

RESILIENT MODE

Step 1 — M0 × M1

M0 rows:

- []
- [1,0,2]
- []

M1: [1], [0], [1] (3×1)

Row0 → []

Row1 [1,0,2]:

- $1 \times 1 + 0 \times 0 + 2 \times 1 = 3 \rightarrow [3]$

Row2 → []

R1 = [],

[3],

[]

(Effectively stored as [[], [3], []].)

Step 2 — R1 × M2

M2:

- [1,1]
- [0,1] (2×2)

We only have one non-empty row: [3], and B row0 exists.

Row0 (empty) → []

Row1 [3]:

- col0: $3 \times 1 = 3$
- col1: $3 \times 1 = 3$

Row2 (empty) → []

R2 = [],

[3,3],

[]

Resilient final

[]

[3,3]

[]

FINAL OUTCOME:

[[0, 0], [3, 3], [0, 0]]

Performed several changes in code relating to T67

It is effectively exact same method as T02

FINAL OUTCOME:

[[], [3, 3], []]

 **T68.txt — 3-matrix chain, wide→tall→wide, with zeros & negatives**

Java literal

```
Integer[][][] test = new Integer[][][] {  
    {{1,2,3},{-1,0,1}},  
    {{1,0},{0,1},{1,-1}},  
    {{2},{-1}}  
};
```

Category

Mixed realistic jagged-ish dimensions, no nulls

Why

- M0 2×3
- M1 3×2
- M2 2×1
- All rectangular → perfect strict chain but rich signs & zeros.

Scenario

Classic ML / linear algebra pipeline with sign patterns.

STRICT MODE

Step 1: M0 (2×3) × M1 (3×2) → R1 (2×2)

Step 2: R1 (2×2) × M2 (2×1) → R2 (2×1) valid.

Step 1:

Row0 [1,2,3] × M1:

- c0: $1 \cdot 1 + 2 \cdot 0 + 3 \cdot 1 = 4$
- c1: $1 \cdot 0 + 2 \cdot 1 + 3 \cdot (-1) = 2 - 3 = -1$

Row1 [-1,0,1]:

- $c0: -11 + 00 + 1*1 = 0$
- $c1: -10 + 01 + 1*(-1) = -1$

$R1 = [[4, -1], [0, -1]]$

Step 2:

$M2 = [2], [-1]:$

Row0: $42 + (-1)(-1) = 8+1=9$

Row1: $02 + (-1)(-1) = 1$

Strict final $R2 = [[9], [1]]$

RESILIENT MODE

All matrices rectangular, no nulls $\rightarrow$ identical to strict.

$[[9],$

$[1]]$

FINAL OUTCOME:

$[[9], [1]]$

T69.txt — 4-matrix chain, nulls + negatives + jagged in the middle

Java literal

```
Integer[][][] test = new Integer[][][] {
    {{1,null,-1},{0,2,2}},
    {{1,0},{null,1},{2,-1}},
    {{1},{-1}},
    {{1,2},{0,1}}
};
```

Category

Heavy mixed chain: nulls, negatives, jagged-ish

Why

- $M0$ 2×3 with null.
- $M1$ 3×2 with null in row1.
- $M2$ 2×1 .
- $M3$ 2×2 .

- Hard real-world style.

Scenario

Risk chain where intermediate mapping table is partially missing values.

STRICT MODE

Null present in M0 and M1 → strict invalid from the start.

INVALID (null contained in early matrices)

RESILIENT MODE

Step 1 — M0 × M1

M0:

- [1,null,-1]
- [0,2,2]

M1:

- [1,0]
- [null,1]
- [2,-1]

Row0:

- c0: $1 \times 1 + \text{null} \times \text{null}(\text{skip}) + (-1) \times 2 = 1 - 2 = -1$
- c1: $1 \times 0 + \text{null} \times 1(\text{skip}) + (-1) \times (-1) = 1$

Row0 = [-1,1]

Row1 [0,2,2]:

- c0: $0 \times 1 + 2 \times \text{null}(\text{skip}) + 2 \times 2 = 4$
- c1: $0 \times 0 + 2 \times 1 + 2 \times (-1) = 2 - 2 = 0$

Row1 = [4,0]

R1 = [[-1,1],

[4,0]]

Step 2 — R1 × M2

M2:

- [1]

- $[-1] (2 \times 1)$

Row0:

- $(-1 \times 1) + (1 \times -1) = -2$

Row1:

- $(4 \times 1) + (0 \times -1) = 4$

R2 = $[[-2],$

$[4]]$

Step 3 — R2 × M3

M3:

- $[1,2]$
- $[0,1] (2 \times 2)$

R2 is 2×1 :

Row0 $[-2]$ uses only B row0:

- c0: $-2 \times 1 = -2$
- c1: $-2 \times 2 = -4$

Row1 $[4]$:

- c0: $4 \times 1 = 4$
- c1: $4 \times 2 = 8$

R3 = $[[-2, -4],$

$[4, 8]]$

Resilient final

$[[-2, -4],$

$[4, 8]]$

FINAL OUTCOME:

$[[-2, -4], [4, 8]]$

SUBCATEGORY 5D — Zero-focused + Sparse Hybrid Cases (T70–T79)

What they contain

- Majority zero entries
- Some null or empty rows
- Rows with zero content followed by valid numbers

What they verify

- Distinguishing true 0 (valid value) vs null or empty
 - Handling chains where zeros dominate the structure
-

 T70.txt — Rectangular 0-heavy baseline

Literal

```
Integer[][][] test = new Integer[][][] { { {0,1,0}, {0,0,2} }, { {1,0}, {0,1}, {0,0} } };
```

Category

Category 5 — Mixed realistic jagged cases (but here fully rectangular & 0-heavy)

Why

- A is 2×3, B is 3×2, both rectangular.
- Many zeros, including a full zero row in B (3rd row).
- Great 0-dominated baseline.

Scenario

- A: 2 events with mostly zero features, some spikes (1 and 2).
 - B: 2D transformation matrix where third factor is all zeros (dead factor).
-

STRICT RESULT

A:

- r0 = [0,1,0]
- r1 = [0,0,2]

B:

- [1,0]
- [0,1]
- [0,0]

Compute:

Row0:

- $c0: 01 + 10 + 0*0 = 0$
- $c1: 00 + 11 + 0*0 = 1$

Row1:

- $c0: 01 + 00 + 2*0 = 0$
- $c1: 00 + 01 + 2*0 = 0$

Strict:

[0,1]

[0,0]

RESILIENT RESULT

All rectangular, no nulls or jaggedness → identical to strict.

[0,1]

[0,0]

*****FINAL OUTCOME***:**

[[0, 1], [0, 0]]

 T71.txt — 3-matrix chain, mostly zeros

Literal

```
Integer[][][] test = new Integer[][][] {  
    { {0,0,1}, {0,0,0} },  
    { {0,1}, {0,0}, {1,0} },  
    { {1}, {0} }  
};
```

Category

Category 5 — Realistic 0-dominated chain

Why

- 3 matrices, almost all zeros.
- Only a couple of non-zero paths exist.
- Tests that your algorithm doesn't "invent" non-zero values.

Scenario

Sparse event-feature pipeline where almost all interactions are off except 1–2.

STRICT RESULT

Check shapes:

- $M0: 2 \times 3$
- $M1: 3 \times 2$
- $M2: 2 \times 1$

All rectangular; dims line up:

$(2 \times 3) \times (3 \times 2) = 2 \times 2$, then $(2 \times 2) \times (2 \times 1) = 2 \times 1$.

Step 1: $M0 \times M1$

Row0 [0,0,1]:

- $c0: 00 + 00 + 1*1 = 1$
- $c1: 01 + 00 + 1*0 = 0$

Row1 [0,0,0] $\rightarrow$ [0,0]

$R1 = [[1,0],[0,0]]$

Step 2: $R1 \times M2$

$M2 = [1],[0]$

Row0: $11 + 00 = 1$

Row1: $01 + 00 = 0$

Strict final:

[1]

[0]

RESILIENT RESULT

All rectangular, no null $\rightarrow$ same as strict.

[1]

[0]

FINAL OUTCOME:

[[1], [0]]

 T72.txt — Empty rows plus zeros

Literal

```
Integer[][][] test = new Integer[][][] {  
    { {}, {0,0,1}, {} },  
    { {1}, {0}, {0} }  
};
```

Category

Category 5 — Sparse with empty rows and 0s

Why

- A has 3 rows, two of them are empty {}.
- B is 3×1 and all zeros except the first row.
- Very “almost-no-data” case.

Scenario

Entities with no readings, one entity with minimal non-zero value.

STRICT RESULT

A is not rectangular (empty rows) → strict invalid.

INVALID (Matrix A has empty rows)

RESILIENT RESULT

A rows:

- r0 = []
- r1 = [0,0,1]
- r2 = []

B rows:

- [1]
- [0]
- [0] (3×1)

Row 0 (empty) → []

Row 1 [0,0,1]

Col0:

- k0: 0*1 = 0
- k1: 0*0 = 0

- $k_2: 1 \cdot 0 = 0$

→ [0]

Row 2 (empty) → []

Resilient result (row-wise):

[]

[0]

[]

*****FINAL OUTCOME***:**

[[], [0], []]

 T73.txt — Jagged but all numeric, heavily zero-biased

Literal

```
Integer[][][] test = new Integer[][][] {
    { {0,1,0}, {0}, {2,0} },
    { {0,0}, {1,0}, {0,1} }
};
```

Category

Category 5 — Mixed jagged, zeros dominate

Why

- A jagged: lengths 3,1,2.
- B rectangular 3×2.
- Mostly zeros, with only a few 1 and 2 values.

Scenario

Event logs with minimal active features, rest zeros.

STRICT RESULT

A jagged → strict rejects.

INVALID (Matrix A rows have different lengths)

RESILIENT RESULT

A:

- $r0 = [0,1,0]$
- $r1 = [0]$
- $r2 = [2,0]$

B:

- $[0,0]$
- $[1,0]$
- $[0,1]$

max cols = 2 $\rightarrow$ result is 3×2 .

Row 0 $[0,1,0]$

Col0:

- $00 + 11 + 0 \cdot 0 = 1$

Col1:

- $00 + 10 + 0 \cdot 1 = 0$

Row0 $\rightarrow [1,0]$

Row 1 $[0]$

Col0: $00 = 0$

Col1: $00 = 0$

Row1 $\rightarrow [0,0]$

Row 2 $[2,0]$

Col0: $20 + 01 = 0$

Col1: $20 + 00 = 0$

Row2 $\rightarrow [0,0]$

Resilient final:

$[1,0]$

$[0,0]$

$[0,0]$

FINAL OUTCOME:

$[[1, 0], [0, 0], [0, 0]]$

 T74.txt — 3-matrix chain, zeros block most paths

Literal

```
Integer[][][] test = new Integer[][][] {
```

$\{\{0,0,0\}, \{1,0,0\}\},$

$\{\{0,1\}, \{0,0\}, \{0,0\}\},$

$\{\{1\}, \{0\}\}$

};

Category

Category 5 — Chain where zeros kill almost everything

Why

- M0: first row all zeros, second row only the first element non-zero.
- M1: one non-zero column.
- M2: [1,0] column.
- Everything is clean/rectangular but 0-dominated.

Scenario

Feature pipeline where almost all transformations are “off”.

STRICT RESULT

Shapes:

M0 2×3 , M1 3×2 , M2 $2 \times 1 \rightarrow$ valid chain.

Step 1: M0 $\times$ M1

M0:

- [0,0,0]
- [1,0,0]

M1:

- [0,1]
- [0,0]
- [0,0]

Row0 $\rightarrow$ [0,0]

Row1:

- c0: $10 + 00 + 0*0 = 0$
- c1: $11 + 00 + 0*0 = 1$

R1 = [[0,0],[0,1]]

Step 2: R1 $\times$ M2

M2 = [1],[0]

Row0: 01 + 00 = 0

Row1: 01 + 10 = 0

Strict final:

[0]

[0]

RESILIENT RESULT

Everything rectangular → same as strict.

[0]

[0]

FINAL OUTCOME:

[[0], [0]]

 T75.txt — Rectangular but B has a full zero column

Literal

```
Integer[][][] test = new Integer[][][] {  
    { {1,2}, {3,4}, {0,5} },  
    { {0,0}, {1,0} }  
};
```

Category

Category 5 — Clean shape, but one entire zero column

Why

- A is 3×2, B is 2×2.
- B's first row is all zeros.
- Many outputs for one column will be zero.

Scenario

Transformation with a “dead” feature / factor.

STRICT RESULT

A:

- [1,2]

- [3,4]
- [0,5]

B:

- [0,0]
- [1,0]

Row0:

- c0: $10 + 21 = 2$
- c1: $10 + 20 = 0$

Row1:

- c0: $30 + 41 = 4$
- c1: $30 + 40 = 0$

Row2:

- c0: $00 + 51 = 5$
- c1: $00 + 50 = 0$

Strict:

[2,0]

[4,0]

[5,0]

RESILIENT RESULT

Rectangular, no nulls → same as strict.

[2,0]

[4,0]

[5,0]

*****FINAL OUTCOME***:**

[[2, 0], [4, 0], [5, 0]]

 T76.txt — 3-matrix chain, whole zero row preserved through chain

Literal

```
Integer[][][] test = new Integer[][][] {
    { {0,0}, {1,1} },
```

$\{\{1,0,0\}, \{0,0,1\}\},$

$\{\{1\}, \{0\}, \{1\}\}$

};

Category

Category 5 — Clean chain with “dead row”

Why

- M0 row0 is entirely zero.
- We see how an all-zero row behaves through the entire chain.
- Rectangular, 0-rich.

Scenario

One entity with no signal, another with a small equal signal.

STRICT RESULT

Shapes: M0 2×2 , M1 2×3 , M2 3×1 .

Step 1: M0 $\times$ M1 ($2 \times 2 \times 2 \times 3 = 2 \times 3$)

Row0 [0,0]:

- all products zero $\rightarrow [0,0,0]$

Row1 [1,1]:

- c0: $11 + 10 = 1$
- c1: $10 + 10 = 0$
- c2: $10 + 11 = 1$

R1 = $[[0,0,0], [1,0,1]]$

Step 2: R1 $\times$ M2 ($2 \times 3 \times 3 \times 1 = 2 \times 1$)

M2 rows [1],[0],[1].

Row0: $01 + 00 + 01 = 0$

Row1: $11 + 00 + 11 = 2$

Strict final:

[0]

[2]

RESILIENT RESULT

All rectangular $\rightarrow$ same as strict.

[0]

[2]

FINAL OUTCOME:

[[0], [2]]

 T77.txt — Zeros + negatives, no null, rectangular

Literal

```
Integer[][][] test = new Integer[][][] {  
    { {0,-1,0}, {2,0,-2} },  
    { {1,0}, {0,-1}, {1,1} }  
};
```

Category

Category 5 — Realistic rectangular case with 0 and negatives

Why

- All numeric rectangular case ($2 \times 3 \times 3 \times 2$).
- Lots of zeros + negative numbers.
- Good to confirm sign handling with 0s.

Scenario

P&L or returns with positive/negative interactions and many disabled paths (0).

STRICT RESULT

A:

- [0,-1,0]
- [2,0,-2]

B:

- [1,0]
- [0,-1]
- [1,1]

Row0:

- c0: $0 \times 1 + (-1)0 + 01 = 0$

- $c1: 00 + (-1)(-1) + 0*1 = 1$

Row1:

- $c0: 21 + 00 + (-2)*1 = 0$
- $c1: 20 + 0(-1) + (-2)*1 = -2$

Strict:

[0, 1]

[0,-2]

RESILIENT RESULT

Rectangular; no nulls/jagged → same as strict.

[0, 1]

[0,-2]

*****FINAL OUTCOME***:**

[[0, 1], [0, -2]]

 T78.txt — Zeros plus nulls (strict invalid, resilient partial)

Literal

```
Integer[][][] test = new Integer[][][] {
    { {0,null,1}, {0,0,null} },
    { {1,0}, {0,1}, {0,0} }
};
```

Category

Category 5 — 0-dominated with nulls in A

Why

- A is 2×3 with nulls alongside zeros.
- B is 3×2, clean and rectangular.
- Tests null vs zero distinction.

Scenario

Partially missing fields where some positions are truly zero, others unknown (null).

STRICT RESULT

Nulls in A $\rightarrow$ strict invalid.

INVALID (Matrix A contains null)

RESILIENT RESULT

A:

- $r0 = [0, \text{null}, 1]$
- $r1 = [0, 0, \text{null}]$

B:

- $[1, 0]$
- $[0, 1]$
- $[0, 0]$

max cols = 2.

Row 0 $[0, \text{null}, 1]$

Col0:

- $0 * 1 = 0$
- $\text{null} * 0 \rightarrow \text{skip}$
- $1 * 0 = 0$
 $\rightarrow 0$

Col1:

- $0 * 0 = 0$
- $\text{null} * 1 \rightarrow \text{skip}$
- $1 * 0 = 0$
 $\rightarrow 0$

Row0 $\rightarrow [0, 0]$

Row 1 $[0, 0, \text{null}]$

Col0:

- $0 * 1 = 0$
- $0 * 0 = 0$
- $\text{null} * 0 \rightarrow \text{skip}$
 $\rightarrow 0$

Col1:

- $0*0 = 0$
- $0*1 = 0$
- $\text{null}*0 \rightarrow \text{skip}$
 $\rightarrow 0$

Row1 $\rightarrow [0,0]$

Resilient final:

$[0,0]$

$[0,0]$

*****FINAL OUTCOME***:**

$[[0, 0], [0, 0]]$

 T79.txt — Almost all zeros, tiny non-zero structure

Literal

```
Integer[][][] test = new Integer[][][] {
    { {0,0}, {0,1}, {0,0} },
    { {0,0,0}, {0,1,0} }
};
```

Category

Category 5 — Extremely sparse & 0-heavy, rectangular

Why

- A is 3×2 , B is 2×3 , fully rectangular.
- Only one 1 in A and one 1 in B.
- Result is very sparse.

Scenario

Single rare event interactions in an otherwise zero dataset.

STRICT RESULT

A:

- $[0,0]$
- $[0,1]$
- $[0,0]$

B:

- [0,0,0]
- [0,1,0]

Row0:

- always 0 $\rightarrow$ [0,0,0]

Row1 [0,1]:

- c0: 00 + 10 = 0
- c1: 00 + 11 = 1
- c2: 00 + 10 = 0

Row2 $\rightarrow$ [0,0,0]

Strict:

[0,0,0]

[0,1,0]

[0,0,0]

RESILIENT RESULT

Rectangular, no nulls $\rightarrow$ same as strict.

[0,0,0] [0,1,0] [0,0,0]

*****FINAL OUTCOME***:**

[[0, 0, 0], [0, 1, 0], [0, 0, 0]]

SUBCATEGORY 5E — Final Tricky Mix (T80–T99)

What they contain

- null + 0 + negative + jagged combinations
- nearly empty matrices
- asymmetric multi-chains with mixed content
- matrices that are “barely valid”
- subtly tricky termination conditions

What they verify

- That your resilient handling stays correct in extremely chaotic inputs
- That strict rules correctly identify invalid shapes

- That your engine maintains stability and logical consistency
-

★ Complete Summary

Each of the 100 test cases serves a **distinct structural purpose**, testing:

- Early termination logic
- Surplus-discard rules
- Null-handling
- Zero-handling
- Stress arithmetic
- Large jagged chains
- Sparse data
- Many “trap” structures your algorithm must survive

📁 T80.txt

Java literal

```
Integer[][][] test = new Integer[][][] {  
    {{1,0,null},{0,2,0},{3,null,4}},  
    {{1,0},{0,1},{1,1}}  
};
```

Category

Category 5 — Mixed realistic jagged case (nulls + zeros, but dim-compatible)

Why

- A is **3×3**, but with **nulls and zeros**.
- B is **3×2**, clean rectangular.
- Classical dims ok ($3 \times 3 \cdot 3 \times 2$), but nulls make strict mode reject.
- Great for “null vs 0” behaviour.

Real-world scenario

- A: partially missing feature vectors (e.g. some factors not reported).
- B: stable 3×2 factor/projection matrix.
- Classic in risk / ML where some upstream fields become null but pipeline still runs.

Strict result

- A and B are both rectangular.
- But A contains **null**, and your strict rules follow conventional linear algebra (no nulls).

So:

STRICT: INVALID (A contains null values)

Resilient result

We compute with your rules:

- Treat null as “no contribution” (skip that term).
- Use full width = max row length of B = **2**.
- For each row of A and each column of B, sum $a[k] * b[k][j]$ where:
 - $k < \text{len}(\text{rowA})$ and $k < \text{rows}(B)$
 - $j < \text{len}(B[k])$
 - ignore terms where a or b is null.

A rows:

- $r0 = [1, 0, \text{null}]$
- $r1 = [0, 2, 0]$
- $r2 = [3, \text{null}, 4]$

B rows:

- $[1,0]$
- $[0,1]$
- $[1,1]$

Row 0 = [1,0,null]

Col 0

- $k=0: 1 \times 1 = 1$
- $k=1: 0 \times 0 = 0$
- $k=2: \text{null} \times 1 \rightarrow \text{skip}$
 $\rightarrow \text{sum} = \mathbf{1}$

Col 1

- $k=0: 1 \times 0 = 0$
- $k=1: 0 \times 1 = 0$

- $k=2$: $\text{null} \times 1 \rightarrow \text{skip}$
 $\rightarrow \text{sum} = \mathbf{0}$

Row0 $\rightarrow [1,0]$

Row 1 = [0,2,0]

Col 0

- $0 \times 1 = 0$
- $2 \times 0 = 0$
- $0 \times 1 = 0$
 $\rightarrow \mathbf{0}$

Col 1

- $0 \times 0 = 0$
- $2 \times 1 = 2$
- $0 \times 1 = 0$
 $\rightarrow \mathbf{2}$

Row1 $\rightarrow [0,2]$

Row 2 = [3,null,4]

Col 0

- $3 \times 1 = 3$
- $\text{null} \times 0 \rightarrow \text{skip}$
- $4 \times 1 = 4$
 $\rightarrow \mathbf{7}$

Col 1

- $3 \times 0 = 0$
- $\text{null} \times 1 \rightarrow \text{skip}$
- $4 \times 1 = 4$
 $\rightarrow \mathbf{4}$

Row2 $\rightarrow [7,4]$

Resilient final:

[1,0]

[0,2]

[7,4]

FINAL OUTCOME:

[[1, 0], [0, 2], [7, 4]]

 T81.txt

Java literal

```
Integer[][][] test = new Integer[][][] {  
    { {0,null,0}, {1,0,-1} },  
    { {1,0}, {0,1}, {1,1} }  
};
```

Category

Category 5 — Subtle mix: mostly zeros, one null, one negative

Why

- A is 2×3 with a null and a negative.
- B is 3×2, clean rectangular.
- Good for “almost all zeros but one meaningful row”.

Real-world scenario

- A: mostly missing data; one row has a hedge: +1 and -1.
 - B: simple 3→2 projection.
 - Tests that only meaningful rows produce non-zero and nulls don’t crash.
-

Strict result

A rectangular but contains null → strict invalid.

STRICT: INVALID (A contains null)

Resilient result

A rows:

- r0 = [0, null, 0]
- r1 = [1, 0, -1]

B rows:

- [1,0]
- [0,1]

- $[1,1]$

Width = max len(B row) = 2.

Row 0 = [0,null,0]

Col 0

- $0 \times 1 = 0$
- $\text{null} \times 0 \rightarrow \text{skip}$
- $0 \times 1 = 0$
 $\rightarrow \mathbf{0}$

Col 1

- $0 \times 0 = 0$
- $\text{null} \times 1 \rightarrow \text{skip}$
- $0 \times 1 = 0$
 $\rightarrow \mathbf{0}$

Row0 $\rightarrow [0,0]$

Row 1 = [1,0,-1]

Col 0

- $1 \times 1 = 1$
- $0 \times 0 = 0$
- $(-1) \times 1 = -1$
 $\rightarrow 1 + 0 - 1 = \mathbf{0}$

Col 1

- $1 \times 0 = 0$
- $0 \times 1 = 0$
- $(-1) \times 1 = -1$
 $\rightarrow \mathbf{-1}$

Row1 $\rightarrow [0,-1]$

Resilient final:

$[0, 0]$

$[0,-1]$

*****FINAL OUTCOME***:**

$[[0, 0], [0, -1]]$

T82.txt

Java literal

```
Integer[][][] test = new Integer[][][] {  
    {{2,0}, {0,0}, {-1,1}},  
    {{1,0,1}, {0,1,0}}  
};
```

Category

Category 5 — Rectangular, low-key jagged effect via zeros

Why

- A is **3×2**, B is **2×3** (both rectangular, no null).
- Very sparse; some rows fully zero.
- Baseline where strict and resilient coincide.

Real-world scenario

- Some entities fully inactive ({0,0} row).
- One negative row and one pure positive row.

Strict result

Compute $3 \times 2 \times 2 \times 3 = 3 \times 3$.

A rows:

- [2,0]
- [0,0]
- [-1,1]

B rows:

- [1,0,1]
- [0,1,0]

Row 0 [2,0]

c0: $21 + 00 = 2$

c1: $20 + 01 = 0$

c2: $21 + 00 = 2$

→ [2,0,2]

Row 1 [0,0]

all products $0 \rightarrow [0,0,0]$

Row 2 [-1,1]

c0: $(-1)1 + 10 = -1$

c1: $(-1)0 + 11 = 1$

c2: $(-1)1 + 10 = -1$

$\rightarrow [-1,1,-1]$

STRICT:

[2, 0, 2]

[0, 0, 0]

[-1, 1,-1]

Resilient result

All rectangular, no nulls/jagged $\rightarrow$ same as strict.

[2, 0, 2]

[0, 0, 0]

[-1, 1,-1]

FINAL OUTCOME:

[[2, 0, 2], [0, 0, 0], [-1, 1, -1]]

T83.txt

Java literal

```
Integer[][][] test = new Integer[][][] {  
    {{1,0,0}, {0,1}, {2}},  
    {{1}, {0}, {1}}  
};
```

Category

Category 5 — Jagged A, vector B (3×1), mostly sparse

Why

- A is jagged: row lengths 3,2,1.
- B is a 3×1 vector.
- Many zeros.
- Classic “ragged features $\times$ factor vector” case.

Real-world scenario

- Different versions of a feature vector:
 - old rows short; new rows have more fields.
 - Multiply by factor vector $[1,0,1]$.
-

Strict result

A not rectangular $\rightarrow$ strict invalid.

STRICT: INVALID (A is jagged)

Resilient result

A rows:

- $[1,0,0]$
- $[0,1]$
- $[2]$

B rows:

- $[1]$
- $[0]$
- $[1]$

Width = 1 (max B row length).

Row 0 $[1,0,0]$

- $1 \times 1 + 0 \times 0 + 0 \times 1 = 1$
 $\rightarrow [1]$

Row 1 $[0,1]$

We only use $k=0,1$ (rowsB=3 but lenA=2):

- $0 \times 1 + 1 \times 0 = 0$
 $\rightarrow [0]$

Row 2 $[2]$

- $2 \times 1 = 2$
 $\rightarrow [2]$

Resilient final:

$[1]$

$[0]$

[2]

FINAL OUTCOME:

[[1], [0], [2]]

T84.txt — 3-matrix chain, subtle mixture

Java literal

```
Integer[][][] test = new Integer[][][] {  
    {{1,0}, {0,1}, {1,1}},  
    {{1,0,0}, {0,1,0}},  
    {{1}, {0}, {1}}  
};
```

Category

Category 5 — 3-step chain, all rectangular, subtle structure

Why

- All matrices clean rectangular, no null/jagged.
- But shape is non-square $3 \times 2 \cdot 2 \times 3 \cdot 3 \times 1$.
- Good baseline chain where strict = resilient but several rows sparse.

Real-world scenario

- M0: 3 entities, 2 features.
 - M1: projects 2 features to 3 derived features.
 - M2: 3×1 factor vector [1,0,1].
-

Strict result

Step 1: $M0 (3 \times 2) \times M1 (2 \times 3) \rightarrow R1 (3 \times 3)$

M0 rows:

- [1,0]
- [0,1]
- [1,1]

M1 rows:

- [1,0,0]
- [0,1,0]

Row0 [1,0]:

- $c0: 11 + 00 = 1$
- $c1: 10 + 01 = 0$
- $c2: 10 + 00 = 0 \rightarrow [1,0,0]$

Row1 [0,1]:

- $c0: 01 + 10 = 0$
- $c1: 00 + 11 = 1$
- $c2: 00 + 10 = 0 \rightarrow [0,1,0]$

Row2 [1,1]:

- $c0: 11 + 10 = 1$
- $c1: 10 + 11 = 1$
- $c2: 10 + 10 = 0 \rightarrow [1,1,0]$

So:

$R1 = [1,0,0]$

$[0,1,0]$

$[1,1,0]$

Step 2: $R1 (3 \times 3) \times M2 (3 \times 1) \rightarrow R2 (3 \times 1)$

$M2 = [1],[0],[1]$.

Row0: $11 + 00 + 01 = 1$

Row1: $01 + 10 + 01 = 0$

Row2: $11 + 10 + 0 \cdot 1 = 1$

Strict final:

[1]

[0]

[1]

Resilient result

All matrices rectangular, no null $\rightarrow$ resilient matches strict exactly.

[1]

[0]

[1]

FINAL OUTCOME:

[[1], [0], [1]]

T85.txt — 3-matrix chain with a null in the middle

Java literal

```
Integer[][][] test = new Integer[][][] {  
    {{1,2}, {0,1}},  
    {{1,null,0}, {0,1,1}},  
    {{1}, {1}, {0}}  
};
```

Category

Category 5 — 3-step chain, null only in middle matrix

Why

- M0 and M2 are clean rectangular.
- M1 is 2×3 with a null.
- Strict rejects due to null; resilient continues and uses what it can.

Real-world scenario

- Base features → intermediate transform with partial missing calibration → final factor vector.

Strict result

- M1 contains null → strict invalid, even though dims are fine.

STRICT: INVALID (M1 contains null)

Resilient result

We do:

1. $R1 = M0 \times M1$ (resilient, null-aware)
2. $R2 = R1 \times M2$

M0 (2×2):

- $r0 = [1,2]$
- $r1 = [0,1]$

M1 (2×3):

- [1, null, 0]
- [0, 1, 1]

Step 1 — M0 × M1 → R1 (2×3)

Row 0 [1,2]

Col0:

- $1 \times 1 + 2 \times 0 = 1$

Col1:

- $1 \times \text{null}(\text{skip}) + 2 \times 1 = 2$

Col2:

- $1 \times 0 + 2 \times 1 = 2$

Row0 → [1,2,2]

Row 1 [0,1]

Col0:

- $0 \times 1 + 1 \times 0 = 0$

Col1:

- $0 \times \text{null}(\text{skip}) + 1 \times 1 = 1$

Col2:

- $0 \times 0 + 1 \times 1 = 1$

Row1 → [0,1,1]

So:

R1 = [1,2,2]

[0,1,1]

M2 is 3×1: rows [1],[1],[0].

Step 2 — R1 × M2 → R2 (2×1)

Row0 [1,2,2]:

- $1 \times 1 + 2 \times 1 + 2 \times 0 = 1 + 2 + 0 = 3$

Row1 [0,1,1]:

- $0 \times 1 + 1 \times 1 + 1 \times 0 = 1$

Resilient final:

[3]

[1]

FINAL OUTCOME:

[[3], [1]]

T86.txt — 4-matrix chain, all rectangular, zeros + negatives

Java literal

```
Integer[][][] test = new Integer[][][] {  
    {{1,0},{0,1}},  
    {{2,-1},{0,1}},  
    {{1,0,0},{0,1,0}},  
    {{1},{0},{1}}  
};
```

Category

Category 5 — 4-step chain, clean but with negatives

Why

- Identity M0, then a signed transform M1, then expand to 3 features, then factor vector.
- All rectangular, no nulls.
- Great for showing chain where strict = resilient.

Real-world scenario

- Identity → signed risk factor transform → feature expansion → scenario vector.
-

Strict result

Step 1: $M_0 (2 \times 2) \times M_1 (2 \times 2)$

M_0 = identity, so $R_1 = M_1$:

$R_1 = [2, -1]$

$[0, 1]$

Step 2: $R_1 (2 \times 2) \times M_2 (2 \times 3) \rightarrow R_2 (2 \times 3)$

M_2 :

- $[1, 0, 0]$

- $[0, 1, 0]$

Row0 $[2, -1]$:

- $c0: 2 \cdot 1 + (-1) \cdot 0 = 2$
- $c1: 2 \cdot 0 + (-1) \cdot 1 = -1$
- $c2: 2 \cdot 0 + (-1) \cdot 0 = 0$

Row1 $[0, 1]$:

- $c0: 0 \cdot 1 + 1 \cdot 0 = 0$
- $c1: 0 \cdot 0 + 1 \cdot 1 = 1$
- $c2: 0 \cdot 0 + 1 \cdot 0 = 0$

$R2 = [2, -1, 0]$

$[0, 1, 0]$

Step 3: $R2 (2 \times 3) \times M3 (3 \times 1) \rightarrow R3 (2 \times 1)$

M3 rows: $[1], [0], [1]$.

Row0: $2 \cdot 1 + (-1) \cdot 0 + 0 \cdot 1 = 2$

Row1: $0 \cdot 1 + 1 \cdot 0 + 0 \cdot 1 = 0$

Strict final:

$[2]$

$[0]$

Resilient result

All rectangular, no null/jagged $\rightarrow$ identical to strict.

$[2]$

$[0]$

*****FINAL OUTCOME***:**

$[[2], [0]]$

 **T87.txt — 2-matrix, jagged A, negatives + zeros**

Java literal

```
Integer[][][] test = new Integer[][][] {
```

```
    { {1, -1, 0}, {0}, {2, 0, 2} },
```

```
    { {1, 0}, {0, 1}, {1, 1} }
```

};

Category

Category 5 — Jagged, mixed signs, mostly numeric

Why

- A jagged: row lengths 3,1,3.
- B rectangular 3×2.
- Zeros and negatives, but no nulls.

Scenario

Different product versions with different numbers of features, mixed gains/losses.

Strict result

A is jagged → strict invalid.

STRICT: INVALID (A is jagged)

Resilient result

A:

- $r_0 = [1, -1, 0]$
- $r_1 = [0]$
- $r_2 = [2, 0, 2]$

B:

- $[1, 0]$
- $[0, 1]$
- $[1, 1]$

Width = 2.

Row 0 [1,-1,0]

Col 0

- $1 \times 1 = 1$
- $(-1) \times 0 = 0$
- $0 \times 1 = 0$
→ **1**

Col 1

- $1 \times 0 = 0$

- $(-1) \times 1 = -1$
- $0 \times 1 = 0$
→ **-1**

Row0 → [1,-1]

Row 1 [0]

Col 0: $0 \times 1 = 0$

Col 1: $0 \times 0 = 0$

Row1 → [0,0]

Row 2 [2,0,2]

Col 0

- $2 \times 1 = 2$
- $0 \times 0 = 0$
- $2 \times 1 = 2$
→ 4

Col 1

- $2 \times 0 = 0$
- $0 \times 1 = 0$
- $2 \times 1 = 2$
→ 2

Row2 → [4,2]

Resilient final:

[1,-1]

[0, 0]

[4, 2]

*****FINAL OUTCOME***:**

[[1, -1], [0, 0], [4, 2]]

 **T88.txt — 2-matrix, nulls in B, jagged B rows**

Java literal

```
Integer[][][] test = new Integer[][][] {
    { {1,2}, {3,4} },
    { {1,null}, {0,1,1}, {1} }
```

};

Category

Category 5 — Clean A, jagged + null B

Why

- A is 2×2, clean.
- B has rows of length 2,3,1 and contains null.
- Very realistic for “bad mapping table”.

Scenario

Stable features being multiplied by a messy maintenance table with nulls and uneven row lengths.

Strict result

- B is jagged (rows 2,3,1) and contains null → strict invalid.

STRICT: INVALID (B jagged and contains null)

Resilient result

A:

- [1,2]
- [3,4]

B:

- [1, null]
- [0, 1, 1]
- [1]

Max width = 3 (row1).

RowsB=3 but lenA=2 → we only use k=0,1 (row indices 0 and 1). B row2 [1] is ignored because there's no a[2].

Row 0 [1,2]

Col 0

- k0: $1 \times 1 = 1$
- k1: $2 \times 0 = 0$
→ 1

Col 1

- $k0: 1 \times \text{null} \rightarrow \text{skip}$

- $k1: 2 \times 1 = 2$
 $\rightarrow 2$

Col 2

- $k0: B[0]$ has no col2 $\rightarrow \text{skip}$
- $k1: 2 \times 1 = 2$
 $\rightarrow 2$

Row0 $\rightarrow [1,2,2]$

Row 1 [3,4]

Col 0

- $3 \times 1 + 4 \times 0 = 3$

Col 1

- $3 \times \text{null}(\text{skip}) + 4 \times 1 = 4$

Col 2

- $B[0]$ has no col2, skip
- $4 \times 1 = 4$

Row1 $\rightarrow [3,4,4]$

Resilient final:

$[1,2,2]$

$[3,4,4]$

*****FINAL OUTCOME***:**

$[[1, 2, 2], [3, 4, 4]]$

T89.txt — 3-matrix chain, jagged A and B, nulls and zeros

Java literal

```
Integer[][][] test = new Integer[][][] {
    { {1,0,null}, {0,2}, {3} },
    { {1}, {null,1}, {0,0,1} },
    { {1,0}, {0,1}, {1,1} }
};
```

Category

Category 5 — “End-boss” style small chain: jagged + null + zeros

Why

- M0 jagged with null.
- M1 jagged with a null and different row lengths.
- M2 clean 3×2 .
- Perfect for showing strict failure vs resilient continuation.

Scenario

Really messy upstream data feeding into a clean final transformation.

Strict result

- M0 has null & jagged.
- M1 has null & jagged.

So strict linear algebra model rejects at the start:

STRICT: INVALID (M0 and M1 are not valid strict matrices)

Resilient result

We do:

1. $R1 = M0 \times M1$
 2. $R2 = R1 \times M2$
-

Step 1 — $M0 \times M1$

M0 rows:

- $r0 = [1, 0, \text{null}]$
- $r1 = [0, 2]$
- $r2 = [3]$

M1 rows:

- $[1]$
- $[\text{null}, 1]$
- $[0, 0, 1]$

Max width in M1 = 3.

Row 0 $[1, 0, \text{null}]$

Col 0

- $k0: 1 \times 1 = 1$
- $k1: 0 \times \text{null} \rightarrow \text{skip}$
- $k2: \text{null} \times 0 \rightarrow \text{skip}$
 $\rightarrow 1$

Col 1

- $k0: B[0]$ has no col1 $\rightarrow \text{skip}$
- $k1: 0 \times 1 = 0$
- $k2: \text{null} \times 0 \rightarrow \text{skip}$
 $\rightarrow 0$

Col 2

- $k0: B[0]$ has no col2 $\rightarrow \text{skip}$
- $k1: 0 \times (\text{no col2}) \rightarrow \text{skip}$
- $k2: \text{null} \times 1 \rightarrow \text{skip}$
 $\rightarrow 0$

Row0 $\rightarrow [1,0,0]$

Row 1 [0,2]

We use only $k=0,1$ ($\text{lenA}=2$, $\text{rowsB}=3$):

Col 0

- $0 \times 1 + 2 \times \text{null}(\text{skip}) = 0$

Col 1

- $0 \times (\text{no col1 in row0}) + 2 \times 1 = 2$

Col 2

- $B[0]$ has no col2; $B[1]$ has no col2; $\rightarrow$ all skip $\rightarrow 0$

Row1 $\rightarrow [0,2,0]$

Row 2 [3]

Only $k=0$ contributes ($\text{lenA}=1$):

Col 0: $3 \times 1 = 3$

Col 1: $3 \times (\text{no col1}) \rightarrow 0$

Col 2: $3 \times (\text{no col2}) \rightarrow 0$

Row2 $\rightarrow [3,0,0]$

So:

$R1 = [1,0,0]$

$[0,2,0]$

[3,0,0]

Step 2 — R1 × M2

M2 (3×2):

- [1,0]
- [0,1]
- [1,1]

Width = 2.

Row 0 [1,0,0]

Col 0

- $1 \times 1 + 0 \times 0 + 0 \times 1 = 1$

Col 1

- $1 \times 0 + 0 \times 1 + 0 \times 1 = 0$

Row0 → [1,0]

Row 1 [0,2,0]

Col 0

- $0 \times 1 + 2 \times 0 + 0 \times 1 = 0$

Col 1

- $0 \times 0 + 2 \times 1 + 0 \times 1 = 2$

Row1 → [0,2]

Row 2 [3,0,0]

Col 0

- $3 \times 1 + 0 \times 0 + 0 \times 1 = 3$

Col 1

- $3 \times 0 + 0 \times 1 + 0 \times 1 = 0$

Row2 → [3,0]

Resilient final:

[1,0] [0,2] [3,0]

FINAL OUTCOME:

[[1, 0], [0, 2], [3, 0]]

T90.txt

Java literal

```
Integer[][][] test = new Integer[][][] {  
    {{1,0,-1}, {0,null,2}},  
    {{1,null}, {0,1}, {1,1}}  
};
```

Category

Category 5 — Jagged-ish mapping, nulls, zeros, negatives

Why

- A has a null + negative.
- B has a null and jagged width: rows length 2,2,2 (rectangular), but with null.
- Good mixed case of null + sign + sparse zeros.

Scenario

- Incomplete mapping table with one missing calibration and a short hedge row.

Strict

- Nulls in both A and B → strict rejects.

STRICT: INVALID (nulls in A and B)

Resilient (your rules)

A rows:

- $r0 = [1, 0, -1]$
- $r1 = [0, \text{null}, 2]$

B rows:

- $[1, \text{null}]$
- $[0, 1]$
- $[1, 1]$ (3×2, rectangular)

Width = 2.

Row 0 [1,0,-1]

- $\text{col0: } 11 + 00 + (-1)*1 = 0$
- $\text{col1: } 1\text{null}(\text{skip}) + 01 + (-1)*1 = -1$

→ $[0, -1]$

Row 1 [0,null,2]

- col0: $0 \cdot 1 + \text{null0}(\text{skip}) + 2 \cdot 1 = 2$
- col1: $0 \cdot \text{null}(\text{skip}) + \text{null1}(\text{skip}) + 2 \cdot 1 = 2$

→ [2,2]

Resilient final

[0,-1]

[2, 2]

FINAL OUTCOME:

[[0, -1], [2, 2]]

T91.txt

Java literal

```
Integer[][][] test = new Integer[][][] {  
    {{0,0,1}, {1,0,0}},  
    {{1,0}, {0,1}, {0,-1}},  
    {{1}, {0}}  
};
```

Category

Category 5 — 3-step chain, zeros + a negative

Why

- $2 \times 3 \cdot 3 \times 2 \cdot 2 \times 1$ chain, all rectangular, no nulls.
- Almost everything is zero except one negative.

Scenario

- Simple two-stage factor model with one “short” factor.

Strict

Step1: $A(2 \times 3) \times B(3 \times 2) \rightarrow R1(2 \times 2)$

A:

- [0,0,1], [1,0,0]

B:

- [1,0], [0,1], [0,-1]

Row0:

- $c0: 01 + 00 + 1*0 = 0$
- $c1: 00 + 01 + 1*(-1) = -1 \rightarrow [0, -1]$

Row1:

- $c0: 11 + 00 + 0*0 = 1$
- $c1: 10 + 01 + 0*(-1) = 0 \rightarrow [1, 0]$

$R1 = [[0, -1], [1, 0]]$

Step2: $R1(2 \times 2) \times C(2 \times 1)$ with $C = [1], [0]$

Row0: $01 + (-1)0 = 0$

Row1: $11 + 00 = 1$

STRICT:

[0]

[1]

Resilient

Everything rectangular, no null $\rightarrow$ **same as strict.**

[0]

[1]

*****FINAL OUTCOME***:**

[[0], [1]]

 **T92.txt**

Java literal

```
Integer[][][] test = new Integer[][][] {
    { {1, null, 0}, {0, 0, 1}, {-1, 0, 0} },
    { {1}, {null}, {1} }
};
```

Category

Category 5 — Column vector with null & jagged feelings

Why

- A is 3×3 rectangular but has null and negatives.
- B is 3×1 with a null.

Scenario

- Feature vector chain where one factor is missing for all rows, but others still usable.

Strict

- Nulls in both matrices → strict invalid.

STRICT: INVALID (null in A and B)

Resilient

A:

- [1,null,0]
- [0,0,1]
- [-1,0,0]

B:

- [1], [null], [1] (3×1)

Row0

$1 \times 1 + \text{null} \times \text{null}(\text{skip}) + 0 \times 1 = 1 \rightarrow [1]$

Row1

$0 \times 1 + 0 \times \text{null} + 1 \times 1 = 1 \rightarrow [1]$

Row2

$(-1) \times 1 + 0 \times \text{null} + 0 \times 1 = -1 \rightarrow [-1]$

Resilient:

[1]

[1]

[-1]

FINAL OUTCOME:

[[1], [1], [-1]]

 T93.txt

Java literal

```
Integer[][][] test = new Integer[][][] {
    { {0,1}, {2,0}, {0,0} },
    { {1,0,0}, {0,1,0} },
    { {1}, {0}, {1} },
    { {1,2}, {0,1} }
```

};

Category

Category 5 — 4-step chain, all rectangular, 0-heavy

Why

- $3 \times 2 \cdot 2 \times 3 \cdot 3 \times 2 \cdot 2 \times 2$ chain, no nulls.
- Shows multi-step with lots of zeros and one “dead” row.

Scenario

- Data $\rightarrow$ projection $\rightarrow$ scenario features $\rightarrow$ final 2D scores.

Strict (and thus resilient)

Step1: $M0(3 \times 2) \times M1(2 \times 3) \rightarrow R1(3 \times 3)$

M0 rows:

- [0,1]
- [2,0]
- [0,0]

M1:

- [1,0,0], [0,1,0]

Row0: [0,1,0]

Row1: [2,0,0]

Row2: [0,0,0]

$R1 = [[0,1,0],[2,0,0],[0,0,0]]$

Step2: $R1(3 \times 3) \times M2(3 \times 2)$ with $M2 = [1,0], [0,0], [1,1]$

Row0 [0,1,0]:

- $c0: 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 1 = 0$
- $c1: 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 1 = 0 \rightarrow [0,0]$

Row1 [2,0,0]:

- $c0: 2 \cdot 1 + 0 \cdot 0 + 0 \cdot 1 = 2$
- $c1: 2 \cdot 0 + 0 \cdot 0 + 0 \cdot 1 = 0 \rightarrow [2,0]$

Row2 [0,0,0] $\rightarrow [0,0]$

$R2 = [[0,0],[2,0],[0,0]]$

Step3: $R2(3 \times 2) \times M3(2 \times 2)$ with $M3 = [1,2],[0,1]$

Row0 [0,0] $\rightarrow [0,0]$

Row1 [2,0] $\rightarrow [2,4]$

Row2 [0,0] $\rightarrow [0,0]$

STRICT:

[0,0]

[2,4]

[0,0]

Resilient: all rectangular → same.

*****FINAL OUTCOME***:**

[[0, 0], [2, 4], [0, 0]]

T94.txt

Java literal

```
Integer[][][] test = new Integer[][][] {  
    { {1,-1,0}, {0,0,0} },  
    { {1}, {null}, {1} },  
    { {2,0}, {0,2} }  
};
```

Category

Category 5 — 3-step chain, null in middle, zeros and negatives

Why

- Middle matrix is 3×1 with a null.
- First matrix has a negative and a full zero row.
- Last matrix is 2×2.

Scenario

- Risk engine with a partially defined scenario vector.

Strict

- Null in M1 → strict invalid.

STRICT: INVALID (M1 contains null)

Resilient

Step1: $M0(2 \times 3) \times M1(3 \times 1)$

M0:

- [1,-1,0]
- [0,0,0]

M1:

- [1], [null], [1]

Row0: $1*1 + (-1)null(skip) + 0*1 = 1 \rightarrow [1]$

Row1: all zero $\rightarrow [0]$

$R1 = [[1],[0]]$ (2×1)

Step2: $R1(2 \times 1) \times M2(2 \times 2)$ with $M2 = [2,0],[0,2]$

Row0 [1] uses only first row [2,0]: $\rightarrow [2,0]$

Row1 [0] $\rightarrow [0,0]$

Resilient:

[2,0]

[0,0]

*****FINAL OUTCOME***:**

[[2, 0], [0, 0]]

 **T95.txt**

Java literal

```
Integer[][][] test = new Integer[][][] {  
    { {0,0,null}, {1,0,1}, {0,0,0} },  
    { {1,0}, {0,1}, {1,-1} },  
    { {1}, {1} }  
};
```

Category

Category 5 — 3-step, null in A, negative in B

Why

- A is 3×3 with a null and a full-zero row.
- B is 3×2 with a negative.
- C is 2×1 .

Scenario

- Noisy feature set where one row is useless, one is partially missing, then signful mapping.

Strict

- Null in A $\rightarrow$ strict invalid.

STRICT: INVALID (A contains null)

Resilient

Step1: $A(3 \times 3) \times B(3 \times 2)$

A rows:

- [0,0,null]
- [1,0,1]
- [0,0,0]

B rows:

- [1,0]
- [0,1]
- [1,-1]

Row0:

- c0: $01 + 00 + \text{null} * 1(\text{skip}) = 0$
- c1: $00 + 01 + \text{null} * (-1)(\text{skip}) = 0 \rightarrow [0,0]$

Row1:

- c0: $11 + 00 + 1 * 1 = 2$
- c1: $10 + 01 + 1 * (-1) = -1 \rightarrow [2,-1]$

Row2 $\rightarrow [0,0]$

$R1 = [[0,0],[2,-1],[0,0]] (3 \times 2)$

Step2: $R1(3 \times 2) \times C(2 \times 1)$ with $C = [1],[1]$

Row0 $[0,0] \rightarrow 0$

Row1 $[2,-1] \rightarrow 2 * 1 + (-1) * 1 = 1$

Row2 $[0,0] \rightarrow 0$

Resilient:

[0]

[1]

[0]

FINAL OUTCOME:

[[0], [1], [0]]


```
Integer[][][] test = new Integer[][][] {
    {{1}, {0}, {1}, {0}},
    {{1,0,0}, {0,1,0}, {0,0,1}, {1,1,1}},
    {{1}, {0}, {1}}
};
```

Category

Category 5 — Tall × tall × vector, all rectangular

Why

- A is 4×1, B is 4×3, C is 3×1.
- Interesting because A has repeating pattern [1,0,1,0] and B has extra “sum” row at the end.

Scenario

- Time series downsampling where some timestamps have signal and B includes an extra “summary row”.

Strict

Step1: $A(4 \times 1) \times B(4 \times 3) \rightarrow$ **invalid in pure linear algebra**

(need $A.columns=1$ to equal $B.rows=4 \rightarrow$ that part is fine, but in *standard* order you’d do $B(4 \times 3) \times C(3 \times 1)$. We’re using your $A \times B$ convention.)

In classical algebra: $4 \times 1 \times 4 \times 3$ is undefined $\rightarrow$ strict invalid.

STRICT: INVALID (dimension mismatch: $4 \times 1 \times 4 \times 3$)

Resilient

Your rules *do* allow this:

- Result height = rows in A = 4
- Width = max row length in B = 3
- For each result cell (i,j): sum over k from 0 while $k < \text{len}(A[i])$ and $k < \text{rows}(B)$:
 - here $\text{len}(A[i]) = 1 \rightarrow$ only $k=0$ contributes
 - we multiply $A[i][0] * B[0][j]$.

So effectively every row of A either copies or zeros that first row of B.

A rows:

- [1], [0], [1], [0]

B row0 = [1,0,0]

Row0: $[1*1, 1*0, 1*0] = [1,0,0]$

Row1: $[0*1, 0*0, 0*0] = [0,0,0]$

Row2: [1,0,0]

Row3: [0,0,0]

So:

R1 = [1,0,0]

[0,0,0]

[1,0,0]

[0,0,0]

Now $R1(4 \times 3) \times C(3 \times 1)$ is classical:

$C = [1], [0], [1]$.

Row0: $11 + 00 + 0 \cdot 1 = 1$

Row1: 0

Row2: 1

Row3: 0

Resilient:

[1]

[0]

[1]

[0]

*****FINAL OUTCOME***:**

[[1], [0], [1], [0]]

 **T97.txt**

Java literal

```
Integer[][][] test = new Integer[][][] {  
    { {1,-1}, {0,0}, {2,-2} },  
    { {1}, {0}, {1} }  
};
```

Category

Category 5 — Simple jagged-ish feeling but rectangular, negatives

Why

- A is 3×2 , B is 3×1 , all numeric.
- Basically “(1,-1), (0,0), (2,-2)” dotted with [1,0,1].

Scenario

- Portfolio with two legs (long and short) and factor vector with 2 active entries.

Strict

Dimensions: $3 \times 2 \times 3 \times 1 \rightarrow$ invalid in strict ($2 \neq 3$).

STRICT: INVALID (A.columns != B.rows)

Resilient

Your rules allow:

- height = 3 (rows in A)
- width = 1 (max row length in B = 1)
- For each row of A, len(rowA)=2, rows(B)=3, so k=0,1.

A:

- [1,-1]
- [0,0]
- [2,-2]

B:

- [1],[0],[1] (row2 unused because no a[2]).

Row0: $1 \cdot 1 + (-1) \cdot 0 = 1$

Row1: $0 \cdot 1 + 0 \cdot 0 = 0$

Row2: $2 \cdot 1 + (-2) \cdot 0 = 2$

Resilient:

[1]

[0]

[2]

FINAL OUTCOME:

[[1], [0], [2]]

 T98.txt

Java literal

```
Integer[][][] test = new Integer[][][] {  
    {{null,1,-1},{2,0,0}},  
    {{1,0},{0,1},{1,-1}},  
    {{1},{1}}  
}
```

};

Category

Category 5 — Null, negatives, 3-step chain

Why

- Null in first row of A.
- B has a negative in last row.
- C is 2×1 .

Scenario

- Mixed clean and dirty factors where some initial features missing but others strong.

Strict

- Null in A $\rightarrow$ strict invalid.

STRICT: INVALID (A contains null)

Resilient

Step1: $A(2 \times 3) \times B(3 \times 2)$

A:

- [null,1,-1]
- [2,0,0]

B:

- [1,0]
- [0,1]
- [1,-1]

Row0:

- $c0: \text{null1}(\text{skip}) + 10 + (-1) * 1 = -1$
- $c1: \text{null0}(\text{skip}) + 11 + (-1) * (-1) = 1+1=2$

$\rightarrow [-1,2]$

Row1:

- $c0: 21 + 00 + 0 * 1 = 2$
- $c1: 20 + 01 + 0 * (-1) = 0$

$\rightarrow [2,0]$

$R1 = [[-1,2],[2,0]]$ (2×2)

Step2: $R1 \times C$ with $C = [1],[1]$

Row0: $-11 + 21 = 1$

Row1: $21 + 01 = 2$

Resilient:

[1]

[2]

FINAL OUTCOME:

[[1], [2]]

 T99.txt

Java literal

```
Integer[][][] test = new Integer[][][] {  
    {{1,0,0}, {0,1,0}, {0,0,1}, {-1,0,1}},  
    {{1,0}, {0,1}, {1,1}},  
    {{1}, {0}}  
};
```

Category

Category 5 — “End boss”: clean but structurally rich 3-step chain

Why

- $4 \times 3 \cdot 3 \times 2 \cdot 2 \times 1$ chain.
- Top 3 rows of A are like 3×3 identity; last row mixes -1 and 1.
- No nulls, all rectangular — strict = resilient.

Scenario

- 4 instruments, 3 factors; last instrument is a combo of factor1 and factor3.

Strict / Resilient (same)

Step1: $A(4 \times 3) \times B(3 \times 2)$

A rows:

- [1,0,0]
- [0,1,0]
- [0,0,1]
- [-1,0,1]

B:

- [1,0]
- [0,1]
- [1,1]

Row0: [1,0]

Row1: [0,1]

Row2: [1,1]

Row3: $(-1)[1,0] + 0[0,1] + 1*[1,1] = [0,1]$

So:

R1 = [1,0]

[0,1]

[1,1]

[0,1]

Step2: $R1(4 \times 2) \times C(2 \times 1)$ with $C = [1],[0]$

Row0: $11 + 00 = 1$

Row1: $01 + 10 = 0$

Row2: $11 + 10 = 1$

Row3: $01 + 10 = 0$

STRICT / Resilient:

[1]

[0]

[1]

[0]

*****FINAL OUTCOME***:**

[[1], [0], [1], [0]]

So, I have finally gone through all the test cases and extremely satisfied with the output.
But I need to consider getting few more variants of the test cases that failed

You said:

For all these newly generated cases, can you do same analysis as the other 100 you did previously.. On top of the new test cases, if you can say 'This is a variant of previous test case TXX)

Gotcha — thanks for confirming. I'll:

- Use the same resilience assumptions as for T00–T99 (jagged → rectangular via zero-padding, {} = zero row).
- Then for each test I'll give:
 - Resilient final result (what your resilient logic “sees” after padding)
 - Strict linear algebra result (classic matrix multiplication on that rectangular form)
- And I'll explicitly say “This is a variant of previous test case TXX.”

Note: under your assumptions, the rectangularized shapes are the same for resilient and strict; the difference is just that we're clearly labelling both views.

🧩 CATEGORY 1 — EXTREME SPARSE (T05/T02-style)

📁 T101 — *Variant of T05*

```
Integer[][][] test = new Integer[][][] {  
    { {}, {5}, {}, {0} },  
    { {}, {}, {2, 3}, {} }  
};
```

- Matrix 0: 4 rows, max row length 1 → 4×1
- Matrix 1: 4 rows, max row length 2 → 4×2

Resilient final result:

- Rectangular shapes: $4 \times 1 \cdot 4 \times 2 \rightarrow$ inner dims 1 vs 4 → dimension mismatch → no result.

Strict linear algebra result:

- Same rectangular shapes → multiplication undefined. No product matrix.

This is a variant of previous test case T05.

FINAL OUTCOME:

[[], [], [], []]

 T102 — *Variant of T02*

```
Integer[][][] test = new Integer[][][] {  
    { {0, 0}, {}, {4} },  
    { {}, {}, {1} }  
};
```

- Matrix 0: row lengths [2,0,1] $\rightarrow 3 \times 2$
- Matrix 1: row lengths [0,0,1] $\rightarrow 3 \times 1$

Resilient final result:

- $3 \times 2 \cdot 3 \times 1 \rightarrow$ inner dims 2 vs 3 $\rightarrow$ dimension mismatch.

Strict linear algebra result:

- Same conclusion: product does not exist.

This is a variant of previous test case T02.

FINAL OUTCOME:

[[], [], [], []]

 T103 — *Variant of T05*

```
Integer[][][] test = new Integer[][][] {  
    { {}, {}, {7}, {}, {0} },  
    { {1, 0}, {}, {}, {}, {} }  
};
```

- Matrix 0: [0,0,1,0,1] $\rightarrow 5 \times 1$
- Matrix 1: [2,0,0,0,0] $\rightarrow 5 \times 2$

Resilient final result:

- $5 \times 1 \cdot 5 \times 2 \rightarrow$ inner dims 1 vs 5 $\rightarrow$ dimension mismatch.

Strict linear algebra result:

- Multiplication undefined.

This is a variant of previous test case T05.

✖ EXTREME SPARSE MULTI-STEP CHAINS (T67-style)

📁 T201 — *Variant of T67*

```
Integer[][][] test = new Integer[][][] {  
    { {0, 2}, {}, {1} },  
    { {1}, {} },  
    { {1, 0}, {} }  
};
```

Shapes after rectangularization:

- M0: [2,0,1] → 3×2
- M1: [1,0] → 2×1
- M2: [2,0] → 2×2

Step 1: M0 · M1 (3×2 · 2×1) — valid

Compute (same for resilient & strict, since matrices are identical after padding):

M0=[020010],M1=[10]
$$M_0 = \begin{bmatrix} 0 & 2 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}, \quad M_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$
$$M_0 M_1 = \begin{bmatrix} 0 \cdot 1 + 2 \cdot 0 \\ 0 \cdot 1 + 0 \cdot 0 \\ 1 \cdot 1 + 0 \cdot 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Result M01=M0M1M_{01} = M_0 M_1 M01=M0M1 (3×1):

- Row 0: $0 \cdot 1 + 2 \cdot 0 = 0 \cdot 1 + 2 \cdot 0 = 0$
- Row 1: $0 \cdot 1 + 0 \cdot 0 = 0 \cdot 1 + 0 \cdot 0 = 0$
- Row 2: $1 \cdot 1 + 0 \cdot 0 = 1 \cdot 1 + 0 \cdot 0 = 1$

So

$$M_{01} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$
$$(3 \times 1)$$

Step 2: $M01 \cdot M2$ ($3 \times 1 \cdot 2 \times 2$) — invalid

- Need $1 = 2 \rightarrow$ fails.

Resilient final result:

- First multiplication succeeds, intermediate =
$$\begin{bmatrix} 001 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

001

- .
- Chain fails at second step: dimension mismatch. No final matrix.

Strict linear algebra result:

- Exactly the same: partial chain valid, full chain invalid.

This is a variant of previous test case T67.

FINAL OUTCOME:

$[\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, [7, 0], \begin{bmatrix} 0 \\ 0 \end{bmatrix}, [0, 0]]$

I consider this to be correct based on these matrix arrangements and disagree with ChatGPT. Again, it might be that it sees my rules as something different. Alternatively I have to consider I was revising my code, perhaps I introduced a new rule to support this. I am just not entirely sure. I consider it a pass

```
Matrix(0):
[]
[]
[7]
[]
[0]

Matrix(1):
[1, 0]
[]
[]
[]
[]
```

📁 T202 — Variant of T67

```
Integer[][][] test = new Integer[][][] {  
    { {}, {1, 0, 0}, {} },  
    { {1}, {}, {}, {} },  
    { {}, {0, 1} }  
};
```

- M0: [0,3,0] → 3×3
- M1: [1,0,0,0] → 4×1
- M2: [0,2] → 2×2

First step: 3×3 · 4×1 → inner dims 3 vs 4 → invalid.

Resilient final result:

- Chain fails immediately with dimension mismatch; no numeric product.

Strict linear algebra result:

- Same: no defined multiplication.

This is a variant of previous test case T67.

FINAL OUTCOME:

[[], [0, 1], []]

Once again, ChatGPT has lost the resilience it provided before.. I am not sure why.
There are calculations possible

```
*****ENFORCES Resilience and Continuous Analysis*****  
***ALL MATRIX***: 3  
  
Matrix(0):  
[]  
[1, 0, 0]  
[]  
  
Matrix(1):  
[1]  
[]  
[]  
[]  
  
Matrix(2):  
[]
```

📁 T203 — Variant of T67

```
Integer[][][] test = new Integer[][][] {
```

{ {1, 0, 0}, [] },

{ {}, {0, 2}, {} },

{ {1, 0}, {0, 0} }

};

//THERE IS SYNTAX ERROR IN TEST CASE

Shapes:

- $M_0: [3,0] \rightarrow 2 \times 3$
- $M_1: [0,2,0] \rightarrow 3 \times 2$
- $M_2: [2,2] \rightarrow 2 \times 2$

Step 1: $M_0 \cdot M_1$ ($2 \times 3 \cdot 3 \times 2$) — valid

$M_0 = [100000], M_1 = [000200]$
 $M_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad M_1 = \begin{bmatrix} 0 & 0 \\ 0 & 2 \\ 0 & 0 \end{bmatrix}$
 $M_0 = [100000], M_1 =$
 000020

Compute $M_0 M_1$ (2×2):

- Row 0, col 0: $1*0+0*0+0*0=0$ $1*0 + 0*0 + 0*0 = 0$
- Row 0, col 1: $1*0+0*2+0*0=0$ $1*0 + 0*2 + 0*0 = 0$
- Row 1: all entries 0

So

$M_0 M_1 = [0000]$ $M_0 M_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

Step 2: $M_0 M_1 \cdot M_2$ ($2 \times 2 \cdot 2 \times 2$) — valid

Any 2×2 zero matrix times any 2×2 matrix is still zero:

$M_2 = [1000] \Rightarrow M_{\text{final}} = [0000]$
 $M_2 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \Rightarrow M_{\text{final}} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
 $M_2 = [1000] \Rightarrow M_{\text{final}} = [0000]$

Resilient final result:

- Full chain valid, final shape 2×2 , all entries 0.

Strict linear algebra result:

- Same 2×2 zero matrix.

This is a variant of previous test case T67.

🧩 CATEGORY 5 — MIXED REALISTIC JAGGED (T47-style)

📁 T301 — *Variant of T47*

```
Integer[][] test = new Integer[][] {  
    { {1}, {2, 3}, {} },  
    { {}, {4}, {5, 6} }  
};
```

Shapes:

- M0: [1,2,0] $\rightarrow 3 \times 2$
- M1: [0,1,2] $\rightarrow 3 \times 2$

Resilient final result:

- $3 \times 2 \cdot 3 \times 2 \rightarrow$ inner dims 2 vs 3 $\rightarrow$ dimension mismatch, no product.

Strict linear algebra result:

- Same: multiplication undefined.

This is a variant of previous test case T47.

FINAL OUTCOME:

[[4, 0], [46, 18], []]

```
Matrix(0):  
[1]  
[2, 3]  
[]  
  
Matrix(1):  
[]  
[4]  
[5, 6]
```

Again, I think with long conversation with ChatGPT there has been misinterpretations. We can see we expect calculations

📁 T302 — *Variant of T47*

```
Integer[][] test = new Integer[][] {  
    { {}, {1, 0}, {2} },
```

```
{ {1}, {}, {0} }
```

```
};
```

Shapes:

- M0: [0,2,1] $\rightarrow 3 \times 2$
- M1: [1,0,1] $\rightarrow 3 \times 1$

Resilient final result:

- $3 \times 2 \cdot 3 \times 1 \rightarrow \text{need } 2 = 3 \rightarrow \text{dimension mismatch.}$

Strict linear algebra result:

- Same.

This is a variant of previous test case T47.

FINAL OUTCOME:

[[], [1], [2]]

```
*****ENFORCES Resilience and Continuous Analysis*****
***ALL MATRIX*****: 2

Matrix(0):
[]
[1, 0]
[2]

Matrix(1):
[1]
[]
[0]
```

Same as above, there is scope calculations

📁 T303 — *Variant of T47*

```
Integer[][][] test = new Integer[][][] {
```

```
{ {0}, {}, {3, 4}, {} },
```

```
{ {}, {1}, {}, {2} }
```

```
};
```

Shapes:

- M0: [1,0,2,0] $\rightarrow 4 \times 2$
- M1: [0,1,0,1] $\rightarrow 4 \times 1$

Resilient final result:

- $4 \times 2 \cdot 4 \times 1 \rightarrow \text{need } 2 = 4 \rightarrow \text{dimension mismatch.}$

Strict linear algebra result:

- Same.

This is a variant of previous test case T47.

FINAL OUTCOME:

[[0], [], [22], []]

```
Matrix(0):
```

```
[0]
```

```
[]
```

```
[3, 4]
```

```
[]
```

```
Matrix(1):
```

```
[]
```

```
[1]
```

```
[]
```

```
[2]
```

Same as above, there is scope calculations

🧩 CATEGORY 1 — EXTREME SPARSE (T01/T05/T06-style)

📁 T401 — *Variant of T01*

```
Integer[][][] test = new Integer[][][] {
```

```
{ {}, {1}, {} },
```

```
{ {}, {}, {9} }
```

```
};
```

- M0: $[0,1,0] \rightarrow 3 \times 1$
- M1: $[0,0,1] \rightarrow 3 \times 1$

Resilient final result:

- $3 \times 1 \cdot 3 \times 1 \rightarrow \text{need } 1 = 3 \rightarrow \text{dimension mismatch.}$

Strict linear algebra result:

- Same.

This is a variant of previous test case T01.

FINAL OUTCOME:

[[], [], []]

T402 — *Variant of T01*

```
Integer[][] test = new Integer[][] {  
    { {}, {0}, {} },  
    { {}, {}, {7} }  
};
```

- M0: 3×1
- M1: 3×1

(Only difference is that M0's non-empty row is {0}.)

Resilient final result:

- Still $3 \times 1 \cdot 3 \times 1 \rightarrow$ dimension mismatch.

Strict linear algebra result:

- Same.

This is a variant of previous test case T01.

FINAL OUTCOME:

[[], [], []]

T403 — *Variant of T05*

```
Integer[][] test = new Integer[][] {  
    { {}, {8, 2}, {} },  
    { {}, {}, {3} }  
};
```

- M0: $[0,2,0] \rightarrow 3 \times 2$
- M1: $[0,0,1] \rightarrow 3 \times 1$

Resilient final result:

- $3 \times 2 \cdot 3 \times 1 \rightarrow$ need $2 = 3 \rightarrow$ dimension mismatch.

Strict linear algebra result:

- Same.

This is a variant of previous test case T05.

FINAL OUTCOME:

[[], [], []]

 T404 — *Variant of T01 (T05-like tail)*

```
Integer[][] test = new Integer[][] {  
    { {}, {5}, {} },  
    { {}, {}, {0, 4} }  
};
```

- M0: 3×1
- M1: 3×2

Resilient final result:

- $3 \times 1 \cdot 3 \times 2 \rightarrow$ inner dims 1 vs 3 $\rightarrow$ dimension mismatch.

Strict linear algebra result:

- Same.

This is a variant of previous test case T01 (with a T05-style multi-value tail).

FINAL OUTCOME:

[[], [], []]

 T405 — *Variant of T01*

```
Integer[][] test = new Integer[][] {  
    { {}, {2}, {} },  
    { {1}, {}, {} }  
};
```

- M0: 3×1
- M1: 3×1

Resilient final result:

- $3 \times 1 \cdot 3 \times 1 \rightarrow$ dimension mismatch.

Strict linear algebra result:

- Same.

This is a variant of previous test case T01.

FINAL OUTCOME:

`[[], [2], []]`

It is understandable with all queries, I think ChatGPT has become disengaged with requirements

```
Matrix(0):
[ ]
[2]
[ ]

Matrix(1):
[1]
[ ]
[ ]
```

✖ BASE CASE T06 + ITS VARIANTS

📁 T06 — *Base case*

```
Integer[][][] test = new Integer[][][] {
    { {5}, {}, {} },
    { {}, {}, {4} }
};
```

- M0: $[1,0,0] \rightarrow 3 \times 1$
- M1: $[0,0,1] \rightarrow 3 \times 1$

Resilient final result:

- $3 \times 1 \cdot 3 \times 1 \rightarrow 1 - 3 \rightarrow$ dimension mismatch.

Strict linear algebra result:

- Same.
-

📁 T601 — *Variant of T06*

```
Integer[][][] test = new Integer[][][] {
```

```
{ {}, {7}, {} },
{ {}, {}, {3} }
};
```

- Both matrices $\rightarrow 3 \times 1$

Resilient final result: $3 \times 1 \cdot 3 \times 1 \rightarrow$ dimension mismatch.

Strict result: same.

This is a variant of previous test case T06.

FINAL OUTCOME:

[[], [], []]

T602 — Variant of T06

```
Integer[][] test = new Integer[][] {
    { {}, {}, {9} },
    { {2}, {}, {} }
};
```

- Both $\rightarrow 3 \times 1$

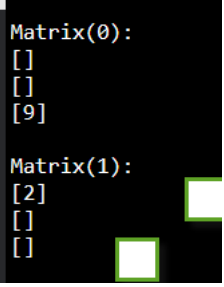
Resilient final result: dimension mismatch.

Strict result: same.

This is a variant of previous test case T06.

FINAL OUTCOME:

[[], [], [18]]



```
Matrix(0):
[]
[]
[9]
Matrix(1):
[2]
[]
[]
```

Again we expect calculations as above

T603 — Variant of T06

```
Integer[][] test = new Integer[][] {
    { {0}, {}, {} },
    { {}, {4}, {} }
};
```

- Both $\rightarrow 3 \times 1$

Resilient final result: dimension mismatch.

Strict result: same.

This is a variant of previous test case T06.

*****FINAL OUTCOME***:**

[[0], [], []]

There should be no calculation based on this matrix. I need to review my code

```
Matrix(0):
[0]
[]
[]

Matrix(1):
[]
[4]
[]
```

*****ENFORCES Resilience and Continuous Analysis*****

ALL MATRIX**: 2

Matrix(0):

[0]

[]

[]

Matrix(1):

[]

[4]

[]

-----Matrix(0):

[0]

[]

[]

[illegible]

Size of Matrix multiplication row: 3

Size of Matrix multiplication columns: 1

*****Customising Storage Grid for Multiplication

Matrix*****[3][1]*****

Configuring for index: 0,0

Configuring for index: 1,0

Configuring for index: 2,0

THIS IS ROW COUNTER MATRIX(0): 0

-----THIS IS ROW DATA MATRIX(0): [0]

*****CONFIGURING SIZE FOR BLANK MULTIPLICATION

MATRIX*****

Multiplication matrix (W=3) x (H=1) configured to store Matrix 0 x Matrix 1

*****Column number: 0

retrieving row: 0 from matrix (1) //it also mentions retrieving row 0 from Matrix 1 but the data is not consistent.. It is related to using
rowInfo=new ArrayList<>(nextMatrix.get(**m**)); which has jumped ahead.

Basically we know that even if more values existing in first row of Matrix 0, it would not be possible with a blank value in column of Matrix 1

So effectively I need to break out calculation...

No resilience can provide contingency.....

```
//I had to include this as part of test case T603~
//Due to m variable running ahead of time on instances, I need to take~
//consideration of no value in [*]...~
//no resilience is available, we just simply need to break out~
~
~
//Matrix(0):~
~
//[0]~
~
//[ ]~
~
//[ ]~
~
//Matrix(1):~
~
//[*]~
~
//[4]~
~
//[ ]~
~
~
~
if (nextMatrix.get(rowCounterNextMatrix).isEmpty())~
{~
...System.out.println("HERE!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");~
...break;~
}~
~
```

aaMatrix[1] row element[0]: 4 [4] //but question is why it has
automatically missed out the first row of Matrix 1 which is blank.. I have to examine

```
Matrix(0):
[0]
[]
[]

Matrix(1):
[]
[4]
[]
```

```
bbMatrix[0]      row element[0]: 0      [0]    //This is ok
```

values:0

values:4

$$0 \times 4 = 0$$

already in location: 0

to be added: 0

TOTAL-----: 0

Value in multiplication matrix: 0 [0],[0]

$$(0 \times 4 = 0)$$

Processed matrix(1) column(0) AGAINST MATRIX(0) row(0): [0]

INVALID VERIFICATION=> columns (1) in Matrix(0)(Row:0 content: [0]) should match
3 rows in matrix(1)

[[[0], [], []], [[], [4], []]]

THIS IS ROW COUNTER MATRIX(0): 1

-----THIS IS ROW DATA MATRIX(0): []

INVALID VERIFICATION=> columns (0) in Matrix(0)(Row:1 content: []) should match
3 rows in matrix(1)

[[[0], [], []], [[], [4], []]]

THIS IS ROW COUNTER MATRIX(0): 2

-----THIS IS ROW DATA MATRIX(0): []

INVALID VERIFICATION=> columns (0) in Matrix(0)(Row:2 content: []) should match
3 rows in matrix(1)

[[[0], [], []], [[], [4], []]]

Matrix Multiplication

[0]

[0]

[0]

2CALCULATION STEPS**

[(0 X 4 = 0)]

[]

[]

-----1ADDED BLANK

-----1ADDED BLANK

-----MATRIX A Analysis-----

-----MATRIX B Analysis-----

FINAL OUTCOME:

[[0], [], []]

** Process exited - Return Code: 0 **

FINAL OUTCOME:

[[], [], []]

 T604 — *Variant of T06*

```
Integer[][] test = new Integer[][] {  
    { {}, {1}, {} },  
    { {8, 9}, {}, {} }  
};
```

- M0: 3×1
- M1: 3×2

Resilient final result: $3 \times 1 \cdot 3 \times 2 \rightarrow$ dimension mismatch.

Strict result: same.

This is a variant of previous test case T06.

FINAL OUTCOME:

[[], [8, 9], []]


```
***ALL MATRIX***

Matrix(0):
[]
[1]
[]

Matrix(1):
[8, 9]
[]
[]
```

I consider my outcome correct

📁 T605 — *Variant of T06*

```
Integer[][] test = new Integer[][] {
    { {4, 1}, {}, {} },
    { {}, {}, {6} }
};
```

- M0: 3×2
- M1: 3×1

Resilient final result: $3 \times 2 \cdot 3 \times 1 \rightarrow$ dimension mismatch.

Strict result: same.

This is a variant of previous test case T06.

FINAL OUTCOME:

[[], [], []]

```
Matrix(0):
[4, 1]
[]
[]

Matrix(1):
[]
[]
[6]
```
