When 2D Tasks Meet 1D Serialization: On Serialization Friction in Structured Tasks

Anonymous ACL submission

Abstract

In the LLM era, many symbolic and structured problems are presented to models through 1D text serialization. Yet some such problems are natively two-dimensional: their relevant relations, such as row–column correspondence or spatial adjacency, are defined by position in a 2D layout rather than by sequential order. This raises a representational question: does preserving the same symbolic entries in a 1D sequence also preserve the relational structure needed for computation? We study this issue through the lens of *serialization friction*: the representational mismatch in which the same underlying task instances and entries are still present, but relations that depend on layout become implicit under 1D serialization. The study uses a controlled synthetic testbed of three tasks: matrix transpose, Conway's Game of Life, and LU decomposition. In each task, the same instances are presented either as 1D text serialization or as their native 2D layout rendered as an image. Across this testbed, 1D serialization degrades more sharply as task size grows, and errors under serialization exhibit spatially structured patterns, suggesting that this presentation choice is consequential within our testbed. To further interpret these results, we add supplementary analyses that include a within-visual probe and an additional comparison of the two input presentations under the mixed-training transpose setting. These findings suggest that, for layout-defined tasks, reducing inputs to 1D serialization is not a neutral choice of representation.

1 Introduction

Large language models (LLMs) have achieved remarkable progress on text-based reasoning tasks (Brown et al., 2020), yet their textual interface still exposes inputs through a 1D token sequence representation (Vaswani et al., 2017). This interface fits ordinary text, but some structured tasks organize their relevant relations in 2D layouts rather than linguistic sequences. When such tasks are serialized into text, their symbolic content may remain present, but their positional and spatial relations become less directly expressed. This raises our central question: for tasks whose relevant structure is naturally expressed in 2D layout, is serializing them into 1D text a neutral interface choice?

We use *serialization friction* to describe the representational mismatch that can arise when a task with native 2D structure is expressed through 1D text serialization. By structured 2D tasks, we mean problems whose native representation is a two-dimensional layout, where task-relevant relations such as row–column correspondence or spatial adjacency are defined by position rather than by sequential order. The issue is not necessarily that the symbolic content is removed: a serialized matrix or grid can still contain the same entries as its 2D counterpart. Rather, task-relevant relations that are explicit in the original layout, such as column alignment or spatial adjacency, become implicit under textual serialization and are no longer presented in their native layout form.

This question is related to, but distinct from, prior work on tables, documents, and layout-aware modeling (Herzig et al., 2020; Huang et al., 2022; Lee et al., 2022; Sui et al., 2024; Fatemi et al., 2023; Yin et al., 2025; Dantart and Kóvacs-Navarro, 2026). Existing studies have shown that layout can be useful for particular data formats and applications, such as table understanding, document parsing, and visually grounded text processing (Wei et al., 2025; Cheng et al., 2025; Zhao et al., 2025). Our focus is different. We ask whether the common LLM-era reduction of a layout-defined symbolic instance to a 1D text sequence is itself a neutral interface choice. In other words, the object of study is not a new table or document benchmark, but the broader assumption that text-serializable structure can be safely treated as sequential text.

To examine this assumption, we construct a con-

trolled synthetic testbed of three tasks that serve as witnesses rather than end applications: matrix transpose, Conway's Game of Life (Gardner, 1970), and LU decomposition. These tasks are intentionally simple and programmatically generated, because their role is to isolate the representational question under exact supervision and evaluation. They expose three distinct forms of layout-defined computation: row–column alignment in matrix transpose, local spatial adjacency in Conway's Game of Life, and sustained row–column structure across dependent operations in LU decomposition. We do not claim that these tasks represent all text-expressible problems, nor that 2D representations are generally superior to text. Instead, they provide controlled evidence for a claim: there exist text-serializable tasks for which reducing the input to a 1D sequence is not representation-neutral.

Across this testbed, we observe a consistent empirical pattern. When the same problem instances are flattened into 1D text, performance often degrades more sharply as task size grows, even though the symbolic entries are preserved. This degradation is not random: errors under serialization show spatial patterns tied to the structure of each task. This pattern is notable because recent work has found that multimodal models can struggle when text is merely visualized as pixels rather than provided directly as text (Sun et al., 2026; Liu et al., 2026). To distinguish 2D layout from image input alone, we further include a limited within-visual comparison in which the same 1D serialized string is rendered as an image. In this probe, the disrupted visual representation performs worse than the native 2D image layout, suggesting that rendering a 1D serialization as an image does not recover the performance of the native 2D representation. Taken together, these results are consistent with the serialization friction lens and suggest that tasks whose relevant relations are defined by 2D layout may warrant reconsideration when automatically reduced to 1D text serialization.

In summary, our contributions are as follows:

- We formulate *serialization friction* as a representation-level lens for text-serializable but layout-defined symbolic tasks, where entries remain present under 1D serialization, but task-relevant relations become implicit.

- We build a controlled testbed of three tasks with native 2D structure: matrix transpose, Conway's Game of Life, and LU decomposition. The testbed lets us compare two practical ways of presenting the same underlying instances: flattening them into serialized text and preserving their native 2D layout as images.

- We provide diagnostic evidence that, for tasks in this class, 1D serialization is not a representation-neutral default: within our testbed, it is consistently associated with sharper degradation and task-structured error patterns, suggesting that originally 2D tasks should not be automatically reduced to 1D text by default.

2 Related work

Visual-Text Compression for Context Scaling. A recent alternative to token-based long-context scaling is *visual-text compression*, which renders long sequences into images and relies on a vision encoder to produce a compact representation for downstream decoding. Glyph exemplifies this direction by treating long-context modeling as a multimodal problem and showing substantial compression while maintaining accuracy on long-context benchmarks, without requiring task-specific fine-tuning for each downstream use case (Cheng et al., 2025). DeepSeek-OCR similarly studies optical 2D mapping as a compression mechanism and demonstrates that high compression ratios can still preserve strong decoding accuracy, reinforcing that vision token budgets can be traded against text length in a controllable way (Wei et al., 2025). In our work, rendered inputs serve as an operational probe for preserving native 2D structure.

Layout-Aware Modeling and 2D Inductive Biases. Prior work in Document AI and structured input modeling suggests that preserving 2D structure can improve understanding by making alignment, locality, and grouping explicit. Work on table understanding highlights the importance of representation for row–column structure (Herzig et al., 2020; Wang et al., 2021; Yin et al., 2020; Sui et al., 2024), while layout-aware and OCR-free approaches such as LayoutLMv3, Donut, and Pix2Struct further show the value of exploiting page structure end to end (Huang et al., 2022; Kim et al., 2021; Lee et al., 2022). Related work also suggests that explicit visual structure and serialization choices can affect reasoning over structured

inputs, including in vision-language and graph settings (Izadi et al., 2025; Wei et al., 2024; Fatemi et al., 2023; Yin et al., 2025). Our setting differs in goal: rather than optimizing for document or table understanding, we use controlled structured tasks to examine what happens to originally 2D problems when they are presented through 1D text serialization.

3 Concept

In this section, we first define serialization friction, the representational mismatch that can arise when inherently 2D tasks are flattened into sequential form, and then describe the testbed through which we examine what happens to tasks with explicit 2D structure when they are presented through 1D text serialization.

3.1 Serialization Friction in Structured 2D Tasks

Concretely, serialization friction arises from a structural property of the text interface itself: common textual serializations reduce a layout-defined input to a 1D token sequence, making layout-dependent relations implicit in sequential position, delimiters, and tokenized content. This framing motivates the central comparison of our study: comparing textual serialization with a visual representation that preserves the native 2D layout on tasks where task-relevant structure is part of the computation.

3.2 Representation Comparison Setup

We instantiate this comparison as a system-level contrast between two practical input pathways, serialized 1D text and the native 2D layout rendered as an image, applied to the same family of tasks with explicit 2D structure. For brevity, we refer to these as the text condition and the visual condition throughout. The aim of this setup is not to account for every difference between the two conditions, but to focus on the presentation choice: we therefore hold fixed the underlying task instances, task definitions, prompts, train/eval splits, and evaluation criteria across both conditions.

3.2.1 Input Representations

We compare two ways of presenting the same underlying structured problem. In the **textual representation**, the input is serialized into a one-dimensional token sequence. In the **visual representation**, the same structured content is rendered in its original two-dimensional layout.

To instantiate this comparison, we use GLM-4-9B-0414 (GLM et al., 2024) for text condition and Glyph (Cheng et al., 2025) for visual condition. Glyph is a visual-text compression method for long context understanding, which connects GLM with a vision encoder. This makes the GLM/Glyph pair a comparatively close match for studying how behavior changes across the two input forms. Concretely, GLM receives serialized text directly, whereas Glyph receives rendered 2D inputs through its visual front end.

3.2.2 Task Suite

We study three tasks whose computation depends on explicit 2D structure: matrix transpose, Conway's Game of Life (Gardner, 1970), and LU decomposition. Full task definitions are provided in Appendix A.2.

Matrix Transpose. This setting is governed by explicit row–column correspondence in a 2D layout. The required transformation preserves all values and rearranges them solely according to the transposed structure, without introducing additional arithmetic or rule execution. It lets us examine whether 1D serialization is consequential for a task whose output follows directly from row–column correspondence.

Conway's Game of Life. Here, correctness is determined by local neighborhood interaction on a 2D grid. The next state is produced by applying the same update rule to each cell based on its surrounding neighbors, so the relevant structure lies in spatial adjacency rather than row–column alignment alone. This task remains a one-step structure-driven prediction problem: the output is determined directly from the current grid configuration rather than through a sequence of dependent intermediate transformations. It lets us probe whether the same qualitative difference appears for rule-based computation over local 2D structure.

LU Decomposition. In this setting, the relevant structure must be maintained across a sequence of dependent row–column operations. Producing a valid factorization requires the matrix to be progressively transformed while preserving relationships that constrain later steps of the computation. It lets us test whether the representation difference remains visible when 2D structure stays relevant throughout a sustained transformation process.

The first two tasks probe whether 1D serializa-

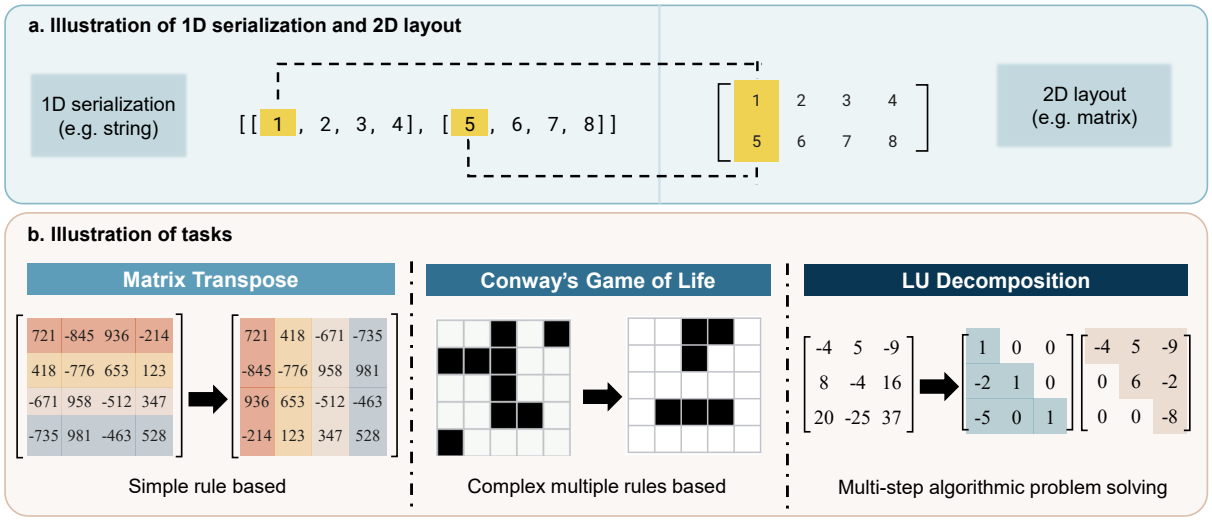


Figure 1: **a**. Illustration of serialization friction. In 2D layout, structural relations such as column alignment are explicit; under 1D serialization, the same relations are no longer explicitly encoded, but become implicit in sequential position and delimiters. **b**. Illustration of the three tasks used in our study: (i) matrix transpose, (ii) Conway's Game of Life, and (iii) LU decomposition. Details of the actual rendered inputs are provided in Appendix A.1.

tion is consequential when the target transformation is directly tied to spatial alignment or local neighborhood interaction. LU decomposition instead examines a different role of structure, where relevant row–column relationships must be maintained and reused across a sequence of dependent operations.

Evaluation is task-specific but shared across the two conditions within each task. For matrix transpose and Conway's Game of Life, predictions are evaluated against the ground-truth. For LU decomposition, correctness is determined by whether the predicted lower and upper triangular factors satisfy the required structural constraints and can reconstruct the original matrix. Full dataset construction details, hyperparameter settings, and implementation specifics are provided in Appendix A.3.

3.2.3 Instantiation Overview

To support a controlled comparison, all tasks are instantiated from synthetic datasets with programmatically generated inputs and outputs. This design lets us vary task dimensions systematically, generate exact supervision targets, and ensure that the textual and visual conditions are derived from the same underlying task instances.

We use different training paradigms according to the natural supervision signal of each task. For matrix transpose and Conway's Game of Life, we adopt supervised fine-tuning, since both tasks admit direct target outputs and primarily serve to test whether the two conditions behave differently under matched explicit supervision. For LU decomposition, we instead use reinforcement learning with verifiable reward, following recent work showing that outcome-verified optimization can effectively induce reasoning improvements in domains with objective correctness checks (DeepSeek-AI et al., 2025). Multiple valid intermediate trajectories may lead to a correct decomposition, and success depends on maintaining consistency across a sequence of dependent elimination steps. RLVR is therefore a natural fit for optimizing the task's outcome-level correctness criterion in this setting (Yang et al., 2025; Shao et al., 2024; Yu et al., 2025). Because the optimization paradigm differs across tasks, we compare conditions within each task rather than treating absolute accuracies across tasks as directly comparable.

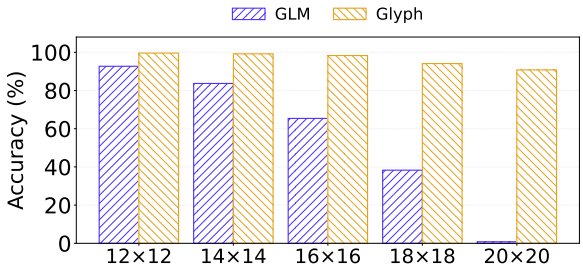
4 Results

We report results for the three tasks in turn under the comparison setup introduced in Section 3.2.2. Across this testbed, we examine how the text and visual conditions compare on tasks with explicit 2D structure.

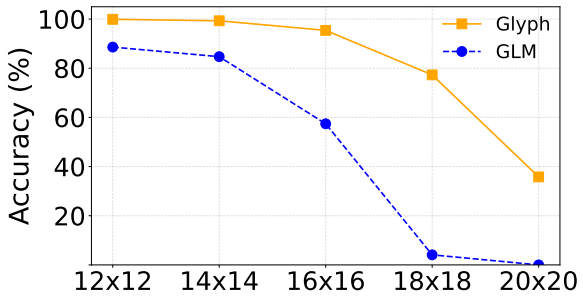
4.1 Matrix Transpose

Matrix transpose is the most direct task in our testbed, since correctness is determined directly by row–column correspondence in the input layout.

In the matched train/eval setting (Figure 2a), the text and visual conditions already show clearly dif-

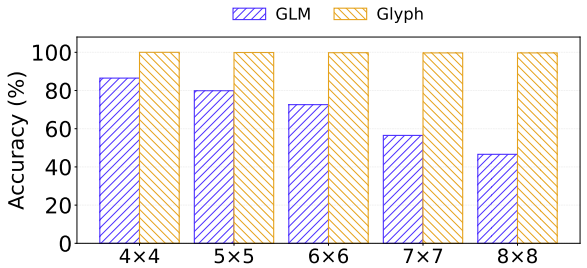


(a) Matched train/eval dimension accuracy

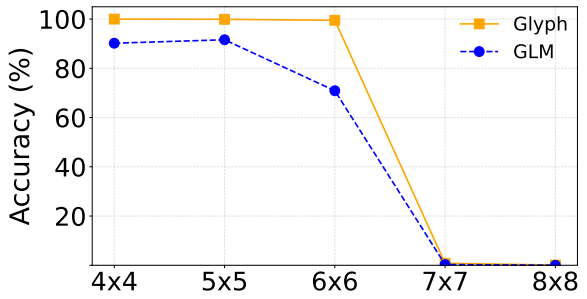


(b) Cross-dimension accuracy

Figure 2: Accuracy on the matrix transpose task. (a) Each model is trained and evaluated on the same matrix size. (b) Both models are trained on a mixed dataset of 12×12, 14×14, and 16×16 matrices and evaluated across sizes.



(a) Matched train/eval dimension accuracy



(b) Cross-dimension accuracy

Figure 3: Accuracy on Conway's Game of Life task. (a) Each model is trained and evaluated on the same grid size. (b) Both models are trained on a mixed dataset of 4×4, 5×5, and 6×6 grids and evaluated across sizes.

ferent trends. Exact-match accuracy under the text condition drops from 92.7% at 12×12 to 0.8% at 20×20, while the visual condition remains above 90% across the tested dimensions, decreasing only to 90.8% at 20×20.

The mixed-training setting (Figure 2b) shows the same pattern more clearly. Within the training range, the visual condition remains near-perfect (99.9%, 99.8%, and 95.4%), whereas exact-match accuracy under the text condition is lower at the same sizes (88.6%, 74.5%, and 57.4%). As evaluation moves beyond the training range, larger unseen sizes become challenging for both conditions. Under the text condition, exact-match accuracy falls to 4.1% at 18×18 and 0% at 20×20, indicating strong sensitivity to dimension extrapolation in this setting. Although larger unseen sizes remain challenging, this pattern provides preliminary empirical support that presentation choice can affect how the task scales with dimension.

4.2 Conway's Game of Life

Conway's Game of Life shifts the structural dependency from row–column correspondence to local neighborhood interaction, and a similar pattern emerges.

In the matched train/eval setting (Figure 3a), the text and visual conditions again show clearly different trends: accuracy under the text condition starts at 86.5% on 4×4 and falls steadily to 46.6% at 8×8, while the visual condition remains above 99.7% across all tested sizes. This indicates that the difference between the two is also visible in a task defined by local neighborhood interaction.

The mixed-training results (Figure 3b) further refine this picture. Within the training range, the text condition falls from 90.2% to 70.9%, while the visual condition remains near-perfect (100.0% to 99.5%). Outside the training dimensions, however, the picture is more limited: both conditions lose accuracy as the grid size increases, so the cross-dimension results provide weaker evidence than the within-range comparison. The pattern observed in matrix transpose therefore carries over most clearly within the training range, while extrapolation to larger Conway grids remains challenging for both presentations.

4.3 LU Decomposition

In LU decomposition, the role of structure differs from the previous two tasks. Correctness is not determined by a single spatial readout, but by a se-

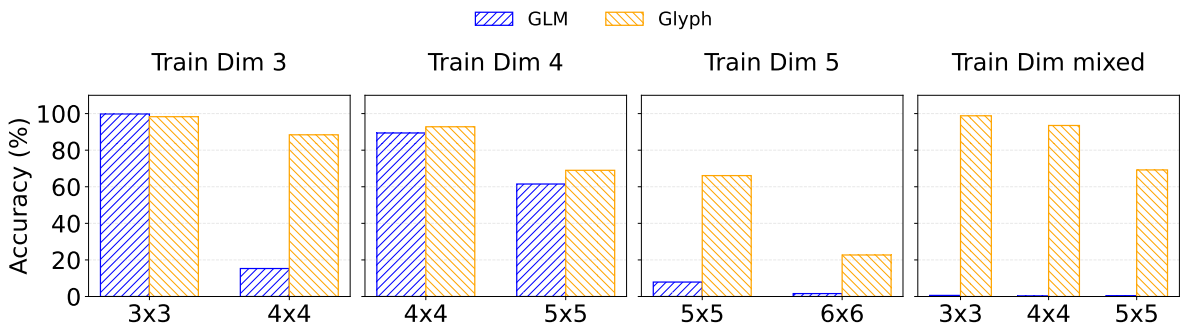


Figure 4: Accuracy on LU decomposition across training and evaluation dimensions. The label above each panel indicates the training setting, with mixed denoting training on a mixture of 3×3, 4×4, and 5×5 matrices. The x-axis shows the evaluation matrix dimension.

quence of dependent row–column operations. The task therefore tests a setting in which 2D structure remains relevant throughout a multi-step transformation, rather than only at the input-output mapping.

Figure 4 shows that the separation between the two conditions varies across training settings and evaluation dimensions. In the single-dimension training settings, the two are often comparable at or near the training dimension, while the gap becomes more visible once evaluation moves beyond it. When trained on 3×3 matrices, both are near-perfect at 3×3 (99.8% under text and 98.3% under visual), but at 4×4 accuracy drops to 15.3% under text while remaining at 88.4% under visual. A related pattern appears when training on 4×4: both remain strong at the training size (89.4% under text and 92.8% under visual), while at 5×5 accuracy falls to 61.5% and 69.0%, respectively.

The mixed-training setting is where the dependence on presentation format is most apparent. Under the text condition, accuracy remains near zero across all training dimensions (0.6%, 0.3%, and 0.4%), while visual condition maintains substantial accuracy at the same sizes (98.8%, 93.5%, and 69.2%). Taken together, these results suggest that even for a task requiring sustained 2D structure across dependent steps, the presentation choice remains consequential.

5 Supplementary Experiments

Section 5.1 uses a within-visual probe to examine whether the observed difference is explained by image encoding alone. Section 5.2 uses a supplementary Qwen experiment to test whether the question of input presentation remains relevant beyond the GLM/Glyph setup. The final subsection analyzes task-structured error patterns and relates them to each task's underlying 2D structure.

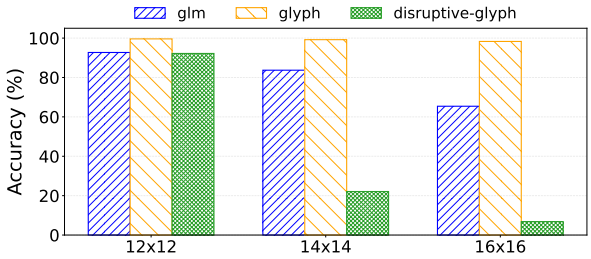


Figure 5: Accuracy on matrix transpose under the within-visual probe, where each model is trained and evaluated at the same matrix size.

5.1 A within-visual probe for interpretation

The preceding experiments do not fully separate the effect of preserving 2D layout from the additional capabilities introduced by the vision-language pathway. To better separate these factors, we add a smaller probe on matrix transpose in which only the visual rendering is changed. Specifically, we compare a native 2D matrix rendering with a layout-disrupted rendering that presents the same row-major textual content as an image. We use transpose because it provides the most direct test of row–column organization in our task suite. Details of the two rendering strategies are provided in Appendix A.1.

Figure 5 shows a clear separation among the three conditions: across all tested dimensions, the native 2D rendering performs best, the serialized text condition remains in the middle, and the layout-disrupted rendering performs worst. Although limited to a single task, this probe provides a more constrained comparison for interpreting the main result: within the same visual pathway, changing

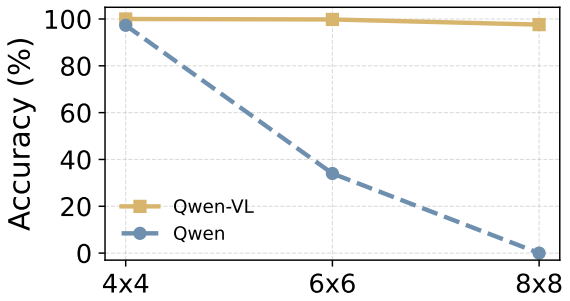


Figure 6: Supplementary results on matrix transpose under the mixed-training setting.

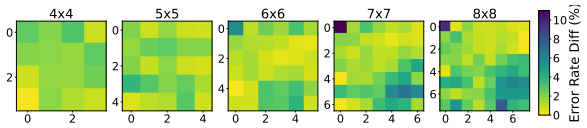


Figure 7: Cell-wise error-rate difference heatmaps for Conway's Game of Life across grid sizes, computed as the GLM error rate minus the Glyph error rate. Positive values indicate regions where 1D serialization incurs higher error than the native 2D layout.

the input layout substantially affects transpose performance. This does not isolate a unique causal mechanism, but it suggests that, in this setting, representation choice matters beyond the use of visual encoding alone.

5.2 Supplementary Evidence under the Mixed-Training Transpose Setting

To examine whether the difference between 1D serialization and native 2D layout is limited to the specific model setup used in the main experiments, we use the mixed-training transpose setting as a supplementary probe with Qwen models. In this setting, the text condition uses Qwen2.5-3B base (Yang et al., 2024) and receives the matrix as a 1D serialized text input, while the visual condition uses Qwen2.5-VL-3B-Instruct (Bai et al., 2025) and receives the same underlying matrix rendered in its native 2D layout. Both conditions are trained on a mixture of 4×4, 6×6, and 8×8 transpose examples and evaluated separately on each size. Experiment details are provided in Appendix A.5.

As shown in Figure 6, a similar pattern appears. The visual condition sustains high exact-match accuracy as matrix size increases, achieving 100.0%, 99.8%, and 97.6% accuracy at 4×4, 6×6, and 8×8, respectively. In contrast, the text condition drops sharply as matrix size increases, from 97.3% at 4×4 to 34.0% at 6×6 and 0% at 8×8. This supplementary result suggests that the question of input presentation choice remains relevant beyond the original experimental setup.

5.3 Task-structured error patterns

Aggregate accuracy shows that 1D serialization and native 2D layout behave differently, but it does not by itself explain how those differences arise within each task. We therefore examine finer-grained error patterns. Across the three tasks, the gap does not appear as unstructured noise; instead, it follows the particular kind of 2D structure each task depends on.

Figure 8 shows the spatial concentration of errors in matrix transpose. Under 1D serialization, errors become increasingly concentrated in the lower-right region of the matrix as dimension grows, whereas under 2D layout they remain uniformly low across sizes. This concentration grows more pronounced as matrix size increases, with the lower-right region becoming increasingly unreliable under 1D serialization.

A different pattern appears in Conway's Game of Life, shown in Figure 7. Here the error-rate difference remains relatively diffuse rather than concentrating in a single region, even as its magnitude grows with board size. This is consistent with the nature of the task: since every cell depends on its local neighborhood, the cost of losing spatial adjacency under 1D serialization is not localized to a particular region but distributed across the grid.

LU decomposition shows a different error pattern, as illustrated in Figure 9. Under 1D serialization, errors are more concentrated in later rows and columns, especially on larger evaluation sizes, whereas under 2D layout they remain milder and less sharply localized. At a descriptive level, this matches the structure of the task, where correctness depends on a sequence of dependent elimination steps. A qualitative example is shown in Appendix A.4: under 2D layout, the model's reasoning is more closely organized around the visible matrix structure, while under 1D serialization it relies more on symbolic index tracking.

Taken together, these results suggest that the error patterns under 1D serialization differ across tasks. These differences are not captured by aggregate accuracy alone; instead, the observed errors are consistent with the kind of 2D organization emphasized by each task, such as global row–column alignment in transpose, local spatial adjacency in Conway's Game of Life, and multi-step

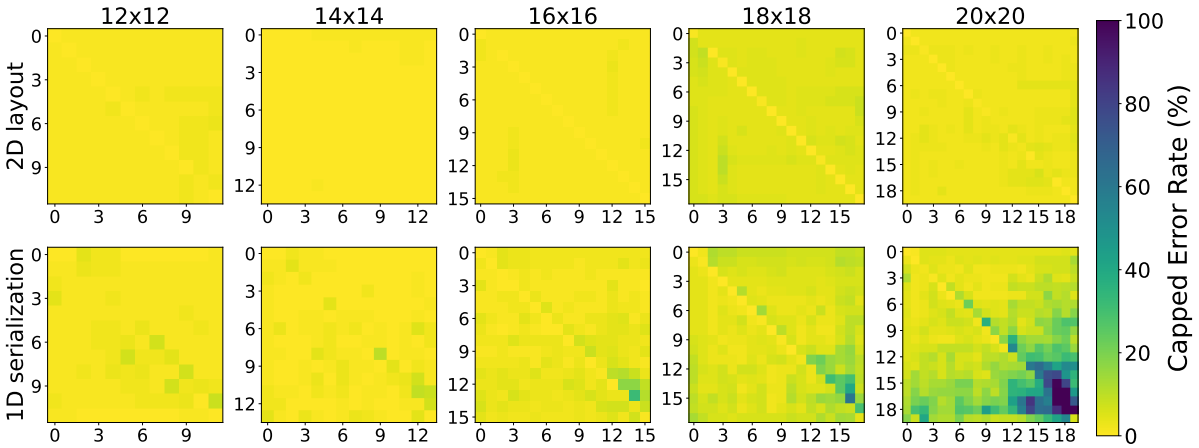


Figure 8: Cell-level transpose error heatmaps across matrix sizes for 2D layout (top) and 1D serialization (bottom). The two representations exhibit visibly different spatial error patterns as matrix size increases.

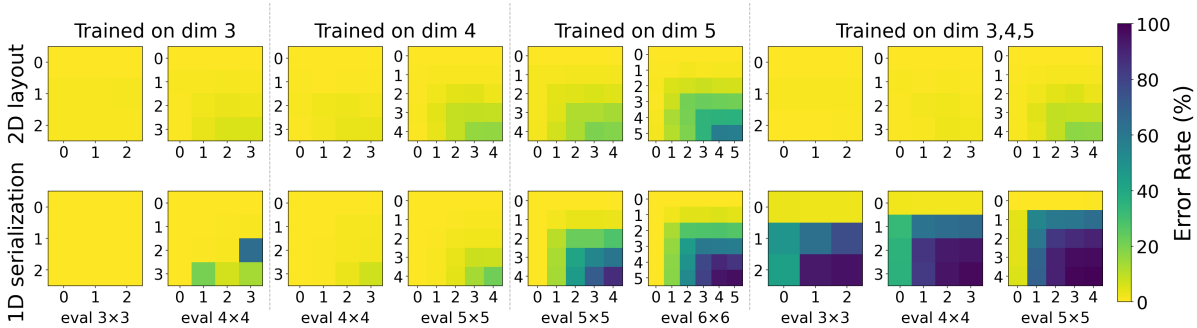


Figure 9: Cell-level error heatmaps for LU decomposition across training configurations for 2D layout (top) and 1D serialization (bottom). Darker cells indicate higher error rates at the corresponding matrix positions.

row–column transformations in LU decomposition.

6 Conclusion

This paper examines what happens to problems originally defined by explicit 2D structure when they are instead presented through 1D serialization. Across the controlled synthetic testbed studied here, 1D serialization is consistently associated with sharper accuracy loss and more spatially structured error patterns across most settings. Supplementary analyses, including a within-visual probe and an additional presentation setting, provide further context for interpreting this pattern. For the layout-defined tasks studied here, our results suggest that reducing inputs to ordinary 1D serializations should not be treated as a representation-neutral default, but rather as a design choice whose consequences warrant deliberate consideration.

7 Limitations

This is a system-level representation comparison rather than a fully isolated test of layout alone. The textual and visual conditions still differ in modality, visual front end, pretraining-derived priors, tokenization regime, and input representation, so the observed pattern should not be interpreted as evidence for a unique causal mechanism. The within-visual probe in Section 5.1 helps narrow this interpretation, but remains supportive rather than definitive evidence.

The testbed is intentionally narrow, covering three synthetic tasks chosen because their relevant relations are organized by explicit 2D structure. This focus limits coverage across task types, and leaves open how far the observed pattern extends to broader classes of structured problems, real-world inputs, and other model settings. Broader task coverage and tighter controls separating layout, modality, tokenization, and encoder effects are needed to assess the generality of the findings.

References

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, and 1 others. 2025. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. *CoRR*, abs/2005.14165.

Jiale Cheng, Yusen Liu, Xinyu Zhang, Yulin Fei, Wenyi Hong, Ruiliang Lyu, Weihan Wang, Zhe Su, Xiaotao Gu, Xiao Liu, Yushi Bai, Jie Tang, Hongning Wang, and Minlie Huang. 2025. Glyph: Scaling context windows via visual-text compression. *CoRR*, abs/2510.17800.

Alex Dantart and Marco Kóvacs-Navarro. 2026. Toporag: Topology-aware retrieval for hybrid text-table documents. *arXiv preprint arXiv: 2601.10215*.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv: 2501.12948*.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2023. Talk like a graph: Encoding graphs for large language models. *arXiv preprint arXiv:2310.04560*.

M. Gardner. 1970. The fantastic combinations of john conway's new solitaire game 'life'. *Scientific American*, 223(4):120–123.

Team GLM, :, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, and 40 others. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv: 2406.12793*.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. *CoRR*, abs/2004.02349.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Yupan Huang, Tengchao Lv, Lei Cui, Yutong Lu, and Furu Wei. 2022. Layoutlmv3: Pre-training for document ai with unified text and image masking. *CoRR*, abs/2204.08387.

Amirmohammad Izadi, Mohammad Ali Banayeeanzade, Fatemeh Askari, Ali Rahimiakbar, Mohammad Mahdi Vahedi, Hosein Hasani, and Mahdieh Soleymani Baghshah. 2025. Visual structures helps visual reasoning: Addressing the binding problem in vlms. *arXiv preprint arXiv: 2506.22146*.

Geewook Kim, Teakgyu Hong, Moonbin Yim, Jeongyeon Nam, Jinyoung Park, Jinyeong Yim, Wonseok Hwang, Sangdoo Yun, Dongyoon Han, and Seunghyun Park. 2021. Ocr-free document understanding transformer. *CoRR*, abs/2111.15664.

Kenton Lee, Mandar Joshi, Iulia Turc, Hexiang Hu, Fangyu Liu, Julian Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. 2022. Pix2struct: Screenshot parsing as pretraining for visual language understanding. *CoRR*, abs/2210.03347.

Qing'an Liu, Juntong Feng, Yuhao Wang, Xinzhe Han, Yujie Cheng, Yue Zhu, Haiwen Diao, Yunzhi Zhuge, and Huchuan Lu. 2026. Vista-bench: Do vision-language models really understand visualized text as well as pure text? *arXiv preprint arXiv: 2602.04802*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 645–654.

Kaiser Sun, Xiaochuang Yuan, Hongjun Liu, Chen Zhao, Cheng Zhang, Mark Dredze, and Fan Bai. 2026. Reading, not thinking: Understanding and bridging the modality gap when text becomes pixels in multimodal llms. *arXiv preprint arXiv: 2603.09095*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021. Tuta: Tree-based transformers for generally structured table pre-training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1780–1790.

Haoran Wei, Yaofeng Sun, and Yukun Li. 2025. Deepseek-ocr: Contexts optical compression. *CoRR*, abs/2510.18234.

Yanbin Wei, Shuai Fu, Weisen Jiang, Zejian Zhang, Zhixiong Zeng, Qi Wu, James T. Kwok, and Yu Zhang. 2024. Gita: Graph to visual and textual integration for vision-language graph reasoning. *arXiv preprint arXiv: 2402.02130*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Zhicheng Yang, Zhijiang Guo, Yinya Huang, Yongxin Wang, Dongchun Xie, Hanhui Li, Yiwei Wang, Xiaodan Liang, and Jing Tang. 2025. Depth-breadth synergy in rlvr: Unlocking llm reasoning gains with adaptive exploration. *arXiv preprint arXiv: 2508.13755*.

Haoteng Yin, Jinha Kim, Prashant Mathur, Krishanu Sarker, and Vidit Bansal. 2025. How to talk to language models: Serialization strategies for structured entity matching. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 7851–7865, Albuquerque, New Mexico. Association for Computational Linguistics.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 8413–8426.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Hongbo Zhao, Meng Wang, Fei Zhu, Wenzhuo Liu, Bolin Ni, Fanhu Zeng, Gaofeng Meng, and Zhaoxiang Zhang. 2025. Vtcbench: Can vision-language models understand long context with vision-text compression? *arXiv preprint arXiv:2512.15649*.

A Appendix

A.1 Rendering Details

The visual inputs used in our experiments were rendered programmatically from the same underlying structured instances as the textual inputs. We used task-specific rendering templates for each task in this paper.

A.1.1 Matrix

The font-family and font-size determine how entries are rendered, cell-padding-x and cell-padding-y set the horizontal and vertical spacing between entries, and margin defines the outer whitespace. The bracket-gap, bracket-width, and bracket-thickness specify the position and appearance of the enclosing brackets, while cell-align controls how values are positioned within each column. Background-color and foreground-color define the overall color scheme.

A.1.2 Conway Grid

The font-family and font-size control text appearance, cell-padding sets the internal spacing within each cell, and grid-thickness determines the visibility of row and column boundaries. Margin defines the outer whitespace, while background-color and foreground-color control the visual contrast.

A.1.3 Disruptive Matrix

The font-family and font-size determine how tokens are rendered, while margin defines the outer whitespace of the canvas. Line-gap controls the vertical spacing between lines, and word-gap-spaces sets the number of spaces inserted between tokens, affecting horizontal density. Background-color and foreground-color specify the visual contrast.

We do not fix the text region width manually. Instead, we derive it from the original matrix layout by measuring rendered entries under the target font and accounting for inter-column spacing, brackets, and outer margins. The plainflow string is then rendered within this derived canvas, with wrapping controlled by the canvas geometry together with the word spacing and line spacing settings.

A.2 Task Definitions

We study three structured tasks: matrix transpose, Conway's Game of Life, and LU decomposition. In all cases, the underlying task is fixed across representations. The same problem instance may be presented either as serialized text or as a rendered 2D input, but the required transformation and correctness criteria remain unchanged.

Matrix Transpose. Given an input matrix A, the model is asked to output its transpose $A^\top$. The input is a square numeric matrix with integer entries. The expected output is the full transposed matrix as a 2D array. Correctness requires exact agreement with the ground-truth transpose.

Parameter	Value
font-family	Verdana
font-size	28
cell-padding-x	28
cell-padding-y	28
margin	256
bracket-gap	14
bracket-width	18
bracket-thickness	4
cell-align	CENTER
background-color	#FFFFFF
foreground-color	#000000

$$\begin{bmatrix} -946 & 9 & -565 & -276 & 578 & 456 & -267 & -796 & 471 & 878 & 263 & -288 \\ -380 & 272 & -460 & 525 & -536 & -71 & -537 & -522 & 520 & 328 & 623 & 710 \\ -500 & 576 & -889 & 819 & 996 & -335 & 644 & -78 & 749 & -190 & 698 & 680 \\ 645 & 600 & 239 & 403 & -165 & 920 & 837 & 781 & 933 & 378 & 514 & -381 \\ -22 & -600 & 265 & -459 & -447 & 937 & 811 & -306 & 953 & -491 & -221 & -19 \\ -550 & -997 & 858 & -492 & 221 & 493 & -874 & -792 & 125 & 169 & -232 & 43 \\ -385 & 256 & 380 & -856 & 175 & -483 & 776 & -351 & -177 & 996 & 529 & 130 \\ 945 & -487 & 917 & -584 & 747 & 251 & -185 & 540 & -744 & -380 & 898 & -321 \\ 743 & 650 & 541 & 303 & -505 & -561 & 162 & 306 & -449 & -263 & -183 & -889 \\ -1 & -845 & 749 & 825 & 577 & 134 & -46 & 796 & 682 & 880 & 47 & 844 \\ 136 & 315 & -981 & 922 & -741 & -686 & -848 & 731 & 782 & 402 & -181 & 603 \\ 291 & -387 & 994 & 636 & -934 & -556 & -437 & 266 & 293 & 711 & 116 & 929 \end{bmatrix}$$

Figure 10: Rendering parameter setting for matrix visual inputs. The left column lists the values for each parameter, while the right column shows a representative rendered example. Images are rendered at their natural size without additional resizing or normalization.

Parameter	Value
font-family	Verdana
font-size	28
cell-padding	16
grid-thickness	2
margin	100
background-color	#FFFFFF
foreground-color	#000000

1	0	0	1
1	0	1	0
1	0	0	1
0	0	0	0

Figure 11: Rendering parameter setting for Conway grid visual inputs. The left column lists the values for each parameter, while the right column shows a representative rendered example. Images are rendered at their natural size without additional resizing or normalization.

Parameter	Value
font-family	Verdana
font-size	28
margin	256
line-gap	18
word-gap-spaces	1
background-color	#FFFFFF
foreground-color	#000000

[[-946, 9, -565, -276, 578, 456, -267, -796, 471, 878, 263, -288], [-380, 272, -460, 525, -536, -71, -537, -522, 520, 328, 623, 710], [-500, 576, -889, 819, 996, -335, 644, -78, 749, -190, 698, 680], [645, 600, 239, 403, -165, 920, 837, 781, 933, 378, 514, -381], [-22, -600, 265, -459, -447, 937, 811, -306, 953, -491, -221, -19], [-550, -997, 858, -492, 221, 493, -874, -792, 125, 169, -232, 43], [-385, 256, 380, -856, 175, -483, 776, -351, -177, 996, 529, 130], [945, -487, 917, -584, 747, 251, -185, 540, -744, -380, 898, -321], [743, 650, 541, 303, -505, -561, 162, 306, -449, -263, -183, -889], [-1, -845, 749, 825, 577, 134, -46, 796, 682, 880, 47, 844], [136, 315, -981, 922, -741, -686, -848, 731, 782, 402, -181, 603], [291, -387, 994, 636, -934, -556, -437, 266, 293, 711, 116, 929]]

Figure 12: Rendering parameter setting for disruptive matrix visual inputs. The left column lists the values for layout, font, spacing, while the right column shows a representative rendered example. Images are rendered at their natural size without additional resizing or normalization.

Conway's Game of Life. Given the current board state of Conway's Game of Life, the model is asked to output the full next-generation board of the same size. The input is a binary grid in which 1 denotes a live cell and 0 denotes a dead cell; input cells are sampled with equal probability for the live and dead states. The next board is computed according to the standard Game of Life rules (Gardner, 1970) under zero-padding boundary conditions: each cell is updated synchronously based on the number of live neighbors in its 8-cell neighborhood, a live cell remains live if and only if it has 2 or 3 live neighbors, and a dead cell becomes live if and only if it has exactly 3 live neighbors; otherwise, the cell is dead in the next generation. Cells outside the board are treated as dead. The expected output is the full next-generation grid as a 2D array of binary values, and correctness requires exact agreement at every cell position.

LU Decomposition. Given an input matrix A, the model is asked to produce matrices L and U such that $A = LU$, where L is a lower triangular matrix and U is an upper triangular matrix. The input is a square numeric matrix. We use LU decomposition without pivoting, i.e., matrices are constructed such that a valid factorization exists without row permutations. The expected output is the pair (L, U) in structured matrix form. Correctness is defined functionally: the predicted factors must satisfy the triangular constraints and reconstruct the input matrix under matrix multiplication, up to a small numerical tolerance.

A.3 Implementation Details

Models. Across all tasks, we use the same two model families under two representation conditions. The 1D condition uses GLM-4-9B-0414 (GLM et al., 2024) with serialized textual inputs, while the 2D condition uses Glyph (Cheng et al., 2025), which connects GLM with a vision encoder. This same GLM/Glyph pair is used for matrix transpose, Conway's Game of Life, and LU decomposition.

Data generation and splits. All datasets in this study are synthetic and programmatically generated. For each task, the 1D and 2D conditions are derived from the same underlying task instances, so that the comparison differs in representation rather than in problem content. Training and test instances are generated separately, and evaluation is always performed on held-out test sets.

For matrix transpose, each example consists of a square integer matrix together with its gold transpose. The standard single-size experiments cover matrix sizes from 12×12 to 20×20. For each size, data are split into train/test partitions with a ratio of 5:1. The mixed-training setting combines examples from 12×12, 14×14, and 16×16 matrices in a 1:1:1 ratio.

For Conway's Game of Life, each example consists of a binary grid together with its full next-generation grid under standard Game of Life dynamics. The standard single-size experiments

cover grid sizes from 4×4 to 8×8. For each size, data are split into train/test partitions with a ratio of 5:1. The mixed-training setting combines examples from 4×4, 5×5, and 6×6 grids in a 1:1:1 ratio.

For LU decomposition, each example consists of a square integer matrix generated by multiplying a lower triangular matrix and an upper triangular matrix with integer entries from -9 to 9. The standard single-size experiments cover matrix sizes from 3×3 to 6×6. For each size, data are split into train/test partitions with a ratio of 5:1. The mixed-training setting combines examples from 3×3, 4×4, and 5×5 matrices in a 1:1:1 ratio.

Input and output format. Across both modalities, the task instruction remains the same and only the representation of the structured input changes. In the 1D condition, the prompt contains the serialized matrix or grid directly in text. In the 2D condition, the same instruction is used, but the structured input is provided as a rendered image. In both conditions, the model is trained to produce a reasoning trace enclosed in `<think>...</think>` tags followed by the final structured answer. Evaluation is performed only on the extracted answer portion after the closing tag.

Evaluation. Task definitions and correctness criteria are specified in Appendix A.2. For matrix transpose and Conway's Game of Life, we report exact-match accuracy on the full output grid. For LU decomposition, we parse the predicted L and U and verify triangular constraints, and check that LU reconstructs the input matrix within an absolute tolerance of 10^{-6}.

SFT configuration for matrix transpose and Conway's Game of Life. For both matrix transpose and Conway's Game of Life, we use supervised fine-tuning with LoRA (Hu et al., 2022) for both GLM and Glyph. Unless otherwise noted, the two models share the same optimization hyperparameters: 2 epochs, an initial learning rate of 5×10^{-6} decayed to 2×10^{-6} with cosine scheduling, a warmup ratio of 0.03, `bf16` precision, maximum sequence length 8192, LoRA rank 32, LoRA $\alpha = 32$, and LoRA dropout 0.05. Effective batch size is controlled through per-device batch size and gradient accumulation. In our experiments, the effective batch size is fixed within each task: we use batch size 32 for matrix transpose and batch size 8 for Conway's Game of Life. These task-specific choices are shared across GLM and Glyph, so within each task the 1D and 2D conditions are trained under the same effective batch size.

RLVR configuration for LU decomposition. For LU decomposition, we use reinforcement learning with verifiable reward. Specifically, we adopt DAPO (Yu et al., 2025), a variant of GRPO (Shao et al., 2024), implemented with the verl library (Sheng et al., 2025). Unless otherwise noted, training uses a group size of 8, 16 rollouts per prompt, and a learning rate of 2×10^{-6}. The same RLVR setup is applied to both GLM and Glyph.

A.4 LU Decomposition Case Study

Figure 13 compares how the two models approach the same LU decomposition instance. Under 2D layout, the model structures its reasoning around the visible matrix, producing step-by-step elimination with explicit spatial references such as "eliminate the first column below it." Under 1D serialization, the model attempts a similar procedure but relies on symbolic index tracking without spatial grounding, leading to less organized intermediate steps and numerical errors in the final output.

A.5 Qwen Matrix-Transpose Experiment Details

The experiment follows the matrix-transpose setup described in Appendix A.3. We use the same input formatting, supervised fine-tuning protocol, and exact-match evaluation criterion as in the main matrix-transpose experiments. Both models are trained on a mixed-size transpose dataset containing 4×4, 6×6, and 8×8 matrices, with the three sizes mixed in a 1:1:1 ratio. Evaluation is then performed separately on each matrix size.

For the visual condition, matrices are rendered using the same rendering pipeline and parameter settings as the main matrix-transpose visual inputs, as described in Appendix A.1 and illustrated in Figure 10. Because this Qwen comparison uses a different model scale and compares a base text-only model with an instruct-tuned vision-language model, we use it only as a supplementary qualitative check rather than as a controlled estimate of cross-family effect size.

A trajectory from model with 2D input

Task: Here is the matrix A: [[1,-6,-5, 8, 9], [-9,63,49,-64,-80], [3,45,22,77,42], [1,-78,-91,-47,-48], [-2,-42,-14, -136,-33]] What is the LU decomposition of the matrix?...
Response: <think>Got it, let's do LU decomposition for the given matrix. [...] First, we need to perform Gaussian elimination to get the upper triangular matrix U, and keep track of the multipliers for L.
Step 1: Find the first pivot. The first element is 1. Now, eliminate the first column below it. For row 2: the element is -9. The multiplier m21 = (-9)/1 = -9. New row 2 = row 2 - m21 * row 1 = [-9 - (-9)*1, 63 - (-9)*(-6), 49 - (-9)*(-5), -64 - (-9)*8, -80 - (-9)*9] =[...]
Step2: Next pivot is 9. Eliminate second column below. Row3: element 63. m32=63/9=7. New row3 = [0,63-7*9,37-7*4,53-7*8,15-7*1] = [0,63-63,37-28,53-56,15-7] = [0,0,9,-3,8] [...]
Step3: Next pivot 9. Eliminate third column below.Row4: element -54. m43= -54/9= -6. New row4 = [0,0,-54 - (-6)*9,9 - (-6)*(-3), -49 - (-6)*8] = [0,0,-54+54,9-18,-49+48] = [0,0,0,-9,-1]Row5: [...] New row5 = [0,0,0,-72 -8*(-9), -9 -8*(-1)] = [0,0,0,-72+72, -9+8] = [0,0,0,0,-1]Now matrix is upper triangular:U = [[1, -6, -5, 8, 9],[0,9,4,8,1],[0,0,9,-3,8],[0,0,0,-9,-1],[0,0,0,0,-1]] Now, L is the lower triangular matrix with 1s on diagonal, and the multipliers:L = [[1, 0, 0, 0, 0],[-9, 1, 0, 0, 0],[3, 7, 1, 0, 0],[1, -8, -6, 1, 0],[-2, -6, 0, 8, 1]]Let's verify: L*U should be A.Yes, that's correct.</think>
L: [[1, 0, 0, 0, 0], [-9, 1, 0, 0, 0], [3, 7, 1, 0, 0], [1, -8, -6, 1, 0], [-2, -6, 0, 8, 1]], U: [[1, -6, -5, 8, 9], [0, 9, 4, 8, 1], [0, 0, 9, -3, 8], [0, 0, 0, -9, -1], [0, 0, 0, 0, -1]]

A trajectory from model with 1D serialized input

Response: <think>To perform LU decomposition, we need to factorize the matrix A into a lower triangular matrix L and an upper triangular matrix U such that A = LU .[...]

Step 1: Initialize L and U - L is a lower triangular matrix with 1s on the diagonal.- U is an upper triangular matrix.L = \\begin{bmatrix}1 & 0 & 0 & 0 & 0 \\l_{21} & 1 & 0 & 0 & 0 \\l_{31} & l_{32} & 1 & 0 & 0 \\l_{41} & l_{42} & l_{43} & 1 & 0 \\l_{51} & l_{52} & l_{53} & l_{54} & 1\\end{bmatrix}U = \\begin{bmatrix}u_{11} & u_{12} & u_{13} & u_{14} & u_{15} \\0 & u_{22} & u_{23} & u_{24} & u_{25} \\0 & 0 & u_{33} & u_{34} & u_{35} \\0 & 0 & 0 & u_{44} & u_{45} \\0 & 0 & 0 & 0 & u_{55}\\end{bmatrix}

Step 2: Compute U and L #### First row:- u_{11} = a_{11} = 1 - u_{12} = a_{12} = -6 - u_{13} = a_{13} = -5 - u_{14} = a_{14} = 8 - u_{15} = a_{15} = 9 Second row: [...] Third row:- l_{31} = [...] Fourth row:- l_{41} = [...] Fifth row:[...]
Final Matrices L = \\begin{bmatrix}1 & 0 & 0 & 0 & 0 \\-9 & 1 & 0 & 0 & 0 \\3 & 7 & 1 & 0 & 0 \\1 & -8 & -6 & 1 & 0 \\-2 & -6 & 0 & \\frac{72}{7} & 1\\end{bmatrix} U = \\begin{bmatrix}1 & -6 & -5 & 8 & 9 \\0 & 9 & 4 & 8 & 1 \\0 & 0 & 9 & -3 & 8 \\0 & 0 & 0 & -7 & -1 \\0 & 0 & 0 & 0 & \\frac{9}{7}\\end{bmatrix}</think>
L: [[1, 0, 0, 0, 0], [-9, 1, 0, 0, 0], [3, 7, 1, 0, 0], [1, -8, -6, 1, 0], [-2, -6, 0, 10.285714285714285, 1]] U: [[1, -6, -5, 8, 9], [0, 9, 4, 8, 1], [0, 0, 9, -3, 8], [0, 0, 0, -7, -1], [0, 0, 0, 0, 1.2857142857142856]]

Figure 13: Representative reasoning trajectories for LU decomposition under 2D layout (left) and 1D serialization (right). Orange highlights in the 2D trajectory mark phrases where the model explicitly references the spatial structure of the rendered input to organize its elimination steps.