BYO-Eval: Build Your Own Dataset for Fine-Grained Visual Assessment of Multimodal Language Models

Ludovic Arnould*
R&D Center, Talan, Paris
ludovic.arnould@talan.com

Salim Khazem*
R&D Center, Talan, Paris

Hugues Ali Mehenni
R&D Center, Talan, Paris

Abstract

Visual Language Models (VLMs) are now sufficiently advanced to support a broad range of applications, including answering complex visual questions, and are increasingly expected to interact with images in varied ways. To evaluate them, current benchmarks often focus on specific domains (e.g., reading charts), constructing datasets of annotated real images paired with pre-defined Multiple Choice Questions (MCQs) to report aggregate accuracy scores. However, such benchmarks entail high annotation costs, risk information leakage, and do not clarify whether failures stem from limitations in visual perception, reasoning, or general knowledge. We propose a new evaluation methodology, inspired by ophthalmologic diagnostics, leveraging procedural generation of synthetic images to obtain control over visual attributes and precisely reveal perception failures in VLMs. Specifically, we build collections of images with gradually more challenging variations in the content of interest (e.g., number of objects in a counting task) while holding other visual parameters constant. This diagnostic allows systematic stress testing and fine-grained failure analysis, shifting the focus from coarse benchmarking toward targeted and interpretable assessment of VLM capabilities. Our code is available at https://github.com/byoeval/BYO-EVAL.

1 Introduction

Vision-language models have achieved remarkable success in a variety of multi-modal tasks [1]. They can generate fluent image descriptions and answer complex visual questions, demonstrating capabilities far beyond early systems [2]. Despite their progress, State-of-The-Art (SoTA) VLMs still show notable shortcomings in basic visual reasoning tasks, such as miscounting objects or incorrectly identifying simple spatial relationships [3, 2], limiting their adoption in industrial contexts [4].

Evaluating these flaws—and VLM performance more broadly—remains difficult, primarily due to the lack of reliable metrics to quantify cross-modal alignment between images, text, and task instructions. Other reasons, including the complexity of the model or the diversity of tasks to evaluate, have led the community to design comprehensive evaluation datasets that provide a one-size-fits-all score, averaging performance across many examples [5, 6].

These benchmarks often aim to globally measure the performance of VLMs against human expectations across numerous use cases [6, 7, 8], assessing high-level abilities of VLMs without precisely distinguishing the visual, textual, or reasoning abilities. Indeed, basic perception weaknesses are masked when looking only at aggregate performance: a model might achieve high overall accuracy on broad benchmarks while consistently stumbling on certain types of questions. The success of the Arena Leaderboard [9] is symptomatic of the lack of standardized criteria or precised, tailored evaluation, as users compare two anonymized Large Language Model (LLM) or VLM outputs based

*Main contributor

Preprint. Under review.

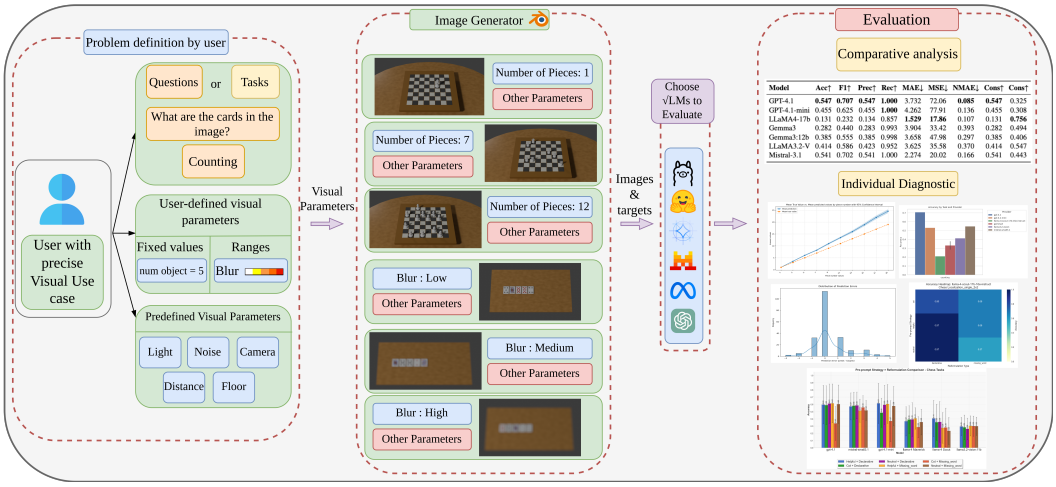


Figure 1: Overview of our evaluation framework. To start, a user indicates in `yaml` configurations the variables to test and their range (e.g., number of objects, blur level) and other fixed values of visual parameters (camera distance, etc.) (first panel). Via Blender Python API (`bpy`), we generate from these input configurations synthetic scenes with their associated legends (middle panel). The resulting images and legends are input into various VLMs (e.g., GPT-4.1, Gemma, ...) for evaluation. Outputs are assessed through multi-dimensional analyses and detailed statistical metrics (right panel).

on a subjective choice of the "best" answer. Moreover, general benchmarks can suffer from several flaws: [10] showed that many visual datasets are corrupted in the sense that VLMs can answer many questions without seeing the image. Moreover, results of proprietary methods are hard to reproduce [11, 12] and benchmarks can be outdated rapidly due to the fast progress of VLMs [13, 14].

To address these shortcomings, we propose a new evaluation framework of VLM that is both diagnostic and task-specific, leveraging procedurally generated synthetic images, using Blender [15], to systematically test them on targeted visual skills. Blender allows us to generate many image variations with precise control over scene attributes—such as object types, counts, and positions—enabling a dynamic, scalable benchmark with built-in ground-truth annotations. Crucially, we design each set of image to evaluate a particular capability of the model, varying the variable of interest while keeping other visual parameters constant, akin to ophthalmologic diagnostics. For example, one task may challenge the model's counting ability by varying the number of objects and asking "How many objects are in the image?", while another task focuses on spatial relations by asking about the relative positions of objects. Our framework thus serves as a diagnostic toolkit: by examining performance on each task in isolation, we can precisely reveal a profile of a VLM's visual abilities, unlike a single aggregate score.

To validate the design and implementation of our evaluation framework, we present an in-depth analysis of several leading VLMs. We generate a diverse set of Poker and Chess images in Blender for a range of evaluation scenarios. Overall, we observe similar limitations in terms of counting, localizing or identifying objects compared to previous work [2, 3, 16, 17], for tasks that are easy for humans. Strikingly, performance varies substantially across domains: models that perform well on chess tasks often fail to generalize to visually similar poker scenes, where occlusion and less rigid layouts are introduced. Such findings support our central claim: general-purpose benchmarks mask brittle behaviors that only emerge under controlled diagnostic stress. We summarize a few key observations regarding the behavior of GPT-4.1 [18] and LLaMA-4-Scout [19] to illustrate the insights of a fine-grained analysis (*cf.* Appendix B):

- **Counting**: GPT-4.1 is very accurate up to 5 pieces but tends to slightly overestimate the count of large object numbers in the Chess dataset, while significantly underestimating the count of Poker cards. Notably, these biases appear to be model-dependent, as LLaMA-4-Scout demonstrates a minor tendency to underestimate object counts in the Chess dataset.

- **Counting with blur**: Both GPT-4.1 and LLaMA-4-Scout exhibit an overestimation pattern in object counting as blur increases, which may be explained by the fact that higher levels of blur can merge pixels, creating the illusion of objects where none exist.
- **Localization of a single Chess piece on a 4x4 board**: This experiment highlights the contextual bias exhibited by the LLaMA-4-Scout model, as it consistently predicts out-of-bounds rows and columns, likely inferring a standard 8x8 chessboard.
- **Localization of a single card on a 3x3 Poker grid**: Both models demonstrate a decline in performance when the element is positioned in the middle column of the poker scene.
- **Relative localization of two Chess pieces on an 8x8 grid**: Both models' performance declines rapidly as the row distance between the two pieces increases, with a clear underestimation of their relative distance.
- **Identification with camera distance (Chess)**: As anticipated, the performance appears to decline as the distance to the chess scene increases.
- **Counting with Horizontal Overlap (Poker cards)**: Both models tend to underestimate the number of cards with horizontal overlap. Additionally, LLaMA-4-Scout seems to particularly struggle with the concept of cards encroaching upon one another, as its underestimation steadily increases with the level of overlap.

Contributions Our work makes the following contributions: (1) We propose a novel evaluation framework for VLM that uses Blender-generated synthetic images to achieve controlled, scalable testing of specific visual skills. (2) We develop a suite of diagnostic tasks (object recognition, counting, localization, etc.) along with an open-source pipeline to generate labeled images and questions for each task. (3) Through extensive experiments on eight SoTA VLMs, we reveal failure modes and fine-grained performance differences between models, providing new insights into the current state of VLM capabilities. (4) Finally, we release our evaluation toolkit publicly, enabling researchers to create new test cases and extend our tasks for probing additional capabilities. We believe this task-specific diagnostic approach will complement existing benchmarks and facilitate the identification of VLM shortcomings in real-life scenarios.

2 Background: evaluation of VLMs

VLMs are generally composed of a visual encoder, a textual decoder and a module to align both modalities in the textual space [20, 1]. The combination of modalities adds an additional layer of complexity towards a precise assessment of performances, as it becomes harder to detect where the model fails. While the training metrics used for foundational VLMs—typically the accuracy of predicting masked tokens—are straightforward, the broad spectrum of inference applications, combined with a widespread adoption beyond deep learning practitioners, has increased the number of tasks to evaluate. This has prompted the development of comprehensive question-answering benchmarks, such as MMMU [21] rather than "low-level" evaluations (e.g., counting objects).

We define **low-level questions** as those focused on basic perceptual tasks—counting, localization, or identification [1]—which form the foundation of image understanding. However, ambiguity may arise: a seemingly low-level question like "How many dogs are in the image?" becomes cross-task if the image includes visually similar objects (e.g., wolves). Conversely, **high-level tasks** combine multiple perceptual abilities and may require reasoning or external knowledge. Examples include "celebrity recognition", "future prediction", "image topic", or "natural relation" [22]. We also classify long-form or cross-task questions (e.g., "What is the time on bottom middle phone?" [23]) as high-level.

High-level benchmarks We refer to [7, 8] for comprehensive surveys and provide a summarized analysis. High-level benchmarks aim to globally measure the performance of VLMs against human expectations across numerous use cases. Images are generally paired with multiple-choice [24] or short-answer questions [25, 23]. They report high-level metrics such as the proportion of correct answers but offer limited insight into each model's visual capabilities and weaknesses. Aggregation of benchmarks [6, 5] further decrease the interpretability of the overall scores. Furthermore, this dynamic reinforces a winner-takes-all logic, where dominant models overshadow alternatives that

[1] Identification could be broken down into more fundamental visual skills like measuring, etc.

may offer better trade-offs in efficiency, domain adaptation, or task specialization. The centralization of attention on a handful of leading models also raises concerns about the reproducibility and transparency of evaluations, as the reported results of leading models can depend on undocumented fine-tuning processes or proprietary datasets that are inaccessible [26, 27, 28, 29].

High-level benchmarks often face limitations that undermine their reliability. As noted by [10], VLMs can achieve over 40% accuracy on multiple-choice benchmarks like MMMU [21] *without* processing the image—due to either data leakage (e.g., test samples seen during training) or questions that can be answered without visual input. Surprisingly, few papers question the bias introduced by the multiple-choice format itself, which may already guides the model toward the correct answer. Moreover, since it's impossible to fully constrain a VLM's answer format, some methods use another LLM to match answers to MCQ choices [24, 22] or compare sequences for open-closed question [30, 31], adding complexity to the method and a potential source of wrong assessment. Finally, the scope of these datasets is limited by the cost of human annotation and their static nature, which can quickly make them obsolete given the rapid progress of VLMs.

Low-level task benchmarks Recent works have highlighted significant challenges in the performance of VLMs concerning tasks that require precise numerical reasoning [32, 33, 34, 35], spatial understanding [17, 16, 36] or basic identification skills [3]. For instance, [17] evaluated GPT-4V on earth observation data and found that, despite the model's proficiency in generating descriptive captions, it exhibited substantial limitations in object counting and localization tasks. Counting benchmarks can contain general [37, 38, 39] or specialized content (e.g., crowd-counting [39], detections from aerial views [40, 17]). However, they mostly provide a basic overview of the model counting abilities (average MAE and accuracy) over images that are too diversified (and that sometimes require reasoning, e.g. "How many giraffes are sitting down?" [37]) to precisely identify failure cases. Most spatial benchmarks focus on spatial reasoning instead of simple localization [16, 41, 36, 42]. SPatial-Eval [16] or SpatialRGPT [42] assess various aspects of spatial reasoning, including understanding relationships, revealing significant shortcomings of state-of-the-art models in tasks requiring detailed spatial comprehension. Regarding low-level identification tasks, [3] show that VLMs struggle to solve seven very specific basic tasks such as spotting overlapping lines or circles.

Related works Most evaluation papers contribute a static dataset with an aggregation score, whereas our work focuses on dataset generation tailored to assessing VLM performance with respect to a few controlled variables. Closer to our work, [2] use synthetic image variations to reveal VLM failures in object detection as distractors increase, but they focus solely on this "binding problem" without offering a systematic data generation or evaluation framework. [43] propose a pipeline that leverages VLMs and text-to-image models [44] to automatically generate descriptions, legends, and questions. However, their approach depends on generative AI, which may lack robustness [45]. [4] manipulate images from standard datasets to generate new tasks and questions using segmentation tools, but this approach lacks fine control over visual parameters. The closest to our work is [46], which generates adaptive datasets from scene graphs using both external images and Blender-rendered objects. However, the task-image link (e.g., for counting) is less explicit due to intermediate representations, loosing control over key visual parameters like lighting or viewpoint. Similarly, [47] automatically build scene graphs with computer vision tools and generate Q&A pairs from existing images, without addressing precise task-level control.

3 Diagnostic methodology and framework

We propose a methodology to obtain a detailed explanation of the performance of a VLM with respect to a given task and visual settings. To this end, we build a set of images with preset levels w.r.t the task at hand, and measure the evolution of the VLM score as the difficulty increases, keeping other parameters constant. Hence, the resulting score evolutions depend only on the difficulty increase. For instance, to test the ability of a VLM to count under blurry conditions, we slightly increase both the number of objects in the image and the level of blur, maintaining the camera view, the location of the objects, etc (as seen in Figure 2). Finally, we retrieve many statistics about the predictions as illustrated in Figure 4.

From a user perspective, our framework enables someone to generate a set of images based on a freely customizable configuration of visual settings 3.1, choose among a set of tasks or questions 3.2,

resulting in a customized image dataset along with diagnostic insights across multiple VLM services (e.g., HuggingFace, Ollama, Groq API) as seen in Figure 4.

3.1 Image generation

We leverage synthetic image generation with Blender to keep full control over the content of the image and legend, in particular over the variables to test and the visual parameters of interest.

Visual parameters Blender's 3D rendering engine allows us to modify a lot of visual parameters (via the python API, `bpy`). In our framework, we have implemented a control over blur, camera distance and angle, table shape and texture, light or resolution (see Appendix D). As we experiment with Poker and Chess images, we also have specific parameters that can vary to change the geometry of the chessboard, the pieces, the cards, etc. These settings are defined in `yaml` configuration files, where a user sets for each parameter either a value or a range of values as shown in Appendix D.

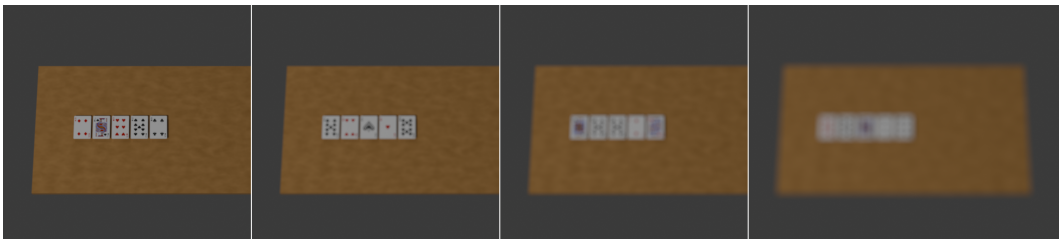


Figure 2: Examples of Poker images from light (left) to very high (right) blur. Images are cropped and overlap for display purposes.

Generation configuration content Our framework emphasizes the difference between the variables of interest, which vary within a predefined range, and the other constant parameters. In detail, consider $n \in \mathbb{N}$ variables $v_1, \dots, v_n$, where each variable v_i can take values either from a discrete set $V_i = \{v_{i,1}, v_{i,2}, \dots, v_{i,k_i}\}$ or from a discrete interval $V_i = [v_{i,\min}, v_{i,\max}]$. To construct our dataset, we systematically generate all possible combinations of these variable values:

$$\mathcal{D} = V_1 \times V_2 \times \cdots \times V_n$$

where $\times$ denotes the Cartesian product. We also duplicate configurations according to the (expected, or measured from a few tries) robustness of the VLM on the given task. For instance, to assess how well a VLM can perceive under blurry conditions, we define one count variable, ranging from 1 to 4, and one blur variable, varying among 5 values from a minor to very blurry image. Then, for each count and each blur level, we generate $k \in \mathbb{N}$ images (k often equals 5 or 10).

Synthetic images Synthetic data has been widely used in AI research to provide controlled environments for model training or evaluation [48, 49, 50, 2, 51]. Regarding evaluation, previous works have demonstrated the advantage and relevance of synthetic datasets in testing object recognition, captioning, and reasoning capabilities in multimodal models [46, 51, 3]. In this context, [46] observed a strong correlation between the results obtained on real and synthetic images. As detailed in Section 4.2, we also observed a very strong correlation between the predictions of the VLMs on real and synthetic images for a sample of 80 images (> 0.9 for 6 models out of 8). We refer the reader to Section 5 of [50] for a review focused on the challenges and methods of computer graphic modeling.

3.2 Evaluation of VLMs

Similarly to our dataset generation, our evaluation process is task-focused. In order to conceive our questions, we first determine low-level tasks to assess the visual abilities of VLMs.

Low-level visual tasks We identify 3 key visual skills to foster the adoption of VLMs within an industrial context: counting, identifying, and locating objects. One benefit of dividing an image analysis based on these tasks is that we can establish reality-grounded metrics, as described below.

- **Counting**. Refers to finding the number of instances of a given object. We measure accuracy, Mean Absolute Error (MAE), Mean Squared Error (MSE) and Normalized MAE (NMAE, which is MAE divided by the target if it is different than 0 or MAE).
- **Localization**. Includes two sub-tasks, absolute and relative localizations. In the first case, a grid is drawn on the image and the VLM must identify in which part of the grid an object is located. As we experiment on chess images, we use the absolute positions of the chessboard considered as a grid. Therefore, we use both accuracy and the count metrics to evaluate this skill by measuring the distance in the grid in both axes:

  $$\mathcal{L}_{\text{LOC}}(p_{\text{real}}, p_{\text{pred}}) := |x_{p_{\text{real}}} - x_{p_{\text{pred}}}| + |y_{p_{\text{real}}} - y_{p_{\text{pred}}}|.$$

  Relative localization involves finding the position of an object w.r.t. another object. Similarly, we rely on both accuracy and the grid to quantify the relative distance.
- **Identification**. We simply ask the VLM to identify an object satisfying a condition. The metrics are all accuracy-based.

Beyond these core metrics, we also report standard classification scores (F1, Precision, Recall). Finally, we define *cross tasks* that combine multiple skills, such as identifying and counting a specific object type. Examples and generation strategies for all tasks are detailed in Section 4.1, with full question templates available in Appendix D.2.

Mitigating the linguistic bias In order to enhance the focus on the perception capacities of a VLM instead of textual or reasoning abilities, we introduce several instruction formats as well as preprompts. Each image can thus be used several times as an input for the VLM with different question prompts. We use either no preprompt ("Neutral"), a "Helpful" preprompt (with general indications about the scene), a "Chain-of-Thought (CoT)" preprompt (to ask the model to first think carefully about the task) and two format instructions, "Declarative" (in the shape of "The answer is:") or "Missing_Word" (details in Appendix D.2). Their effects are analyzed in Section 4.2.

4 Diagnostics of different VLMs on poker and chess images

In order to validate our methodology, we build a dataset of chess and poker images and we diagnose several SoTA VLMs on a few representative configurations. Because industrial use-cases demand mastery of low-level vision tasks, we first sought an environment that exposes such challenges while remaining fully controllable. The clean geometric layouts of Chess and Poker let us generate large numbers of scenes and scale task complexity with precision. This design choice is further motivated by the persistent difficulties that VLMs and LLMs display when vision guides downstream reasoning in games [52]. Finally, the popularity and rigorously defined rules of both games translate into readily available image corpora, making them ideal for reproducible, real-world evaluation.

The full list of tasks is presented in Table 1. For each task and game, we generate a tailored set of images. For example, in counting, the number of pieces ranges from 1 to 20 and cards from 2 to 15, with 5 to 40 repetitions per level depending on the setup. We also vary visual conditions such as blur, camera distance, and object occlusion (e.g., card overlap). Examples are shown in Figures 2 and 3, with additional visuals in Appendix B. Each image is paired with a question, and we compute several metrics from the model responses. Section 4.1 focuses on VLM counting abilities, while Section 4.2 presents a broader benchmark across SoTA models.

4.1 Individual diagnostic

In this section, we conduct extensive analyses of one proprietary model, GPT-4.1 [18] and one open-source model, LLaMA-4-Scout [19]. We challenge the two models on all the aforementioned tasks (Table 1), for both Poker and Chess, using a declarative instruction as well as both a debiased and debiased-CoT preprompt (see Appendix D.2 or the concrete example below). We only report the results of GPT-4.1 on Chess for the counting task (1 to 21 pieces, see Figure 3), with a declarative answer format and a debiased preprompt, which gives the following instruction:

"This is not a real chess game. The number of each piece and their position can vary arbitrary. Just focus on answering the following question based on the visual content. How many pieces are there in the image? The number of pieces in the image is:"

Table 1: Diagnostic tasks, questions, targets, and variables for evaluation of VLM capabilities. We only display the core of the questions; the full content can be found in Appendix D.2.

Task	Question main content	Target	Variable to test
Counting (basic)	How many cards/pieces?	Number of objects	Number of objects
Counting against blur	How many cards/pieces?	Number of objects	Number of objects Blur intensity
Counting and identification	How many objects X?	Number of objects X	Number of objects
Counting against card overlap	How many cards?	Number of cards	Degree of overlap Number of objects
Identification against camera distance	What is the card/piece?	Object class	Camera distance
Absolute localization of a single object	Where is the card/piece?	Object position	Object type, Grid size
Relative localization of two objects (Chess)	Distance between the pieces?	Number of rows/cols	Piece positions

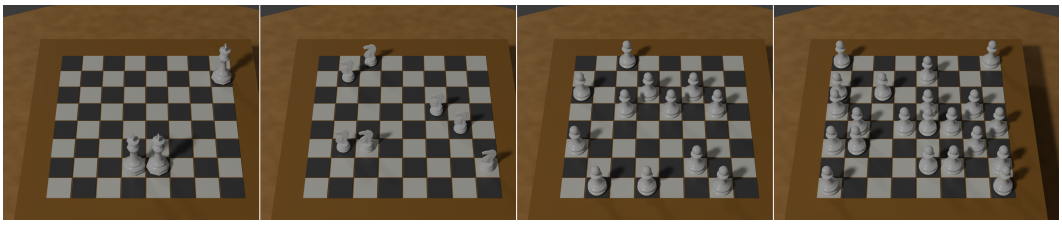


Figure 3: Variations of the number of pieces. Images are cropped and overlap for display purposes.

The key takeaways from this experiment are (1) a clear increase of the MAE w.r.t. the number of pieces on the chessboard and (2) a rather centered distribution of errors, with a slight tendency to overcount (Figure 4). Up to 5 pieces, the model reaches 100% accuracy; then it miscounts one element on average up to 15 pieces, where it starts making more important mistakes.

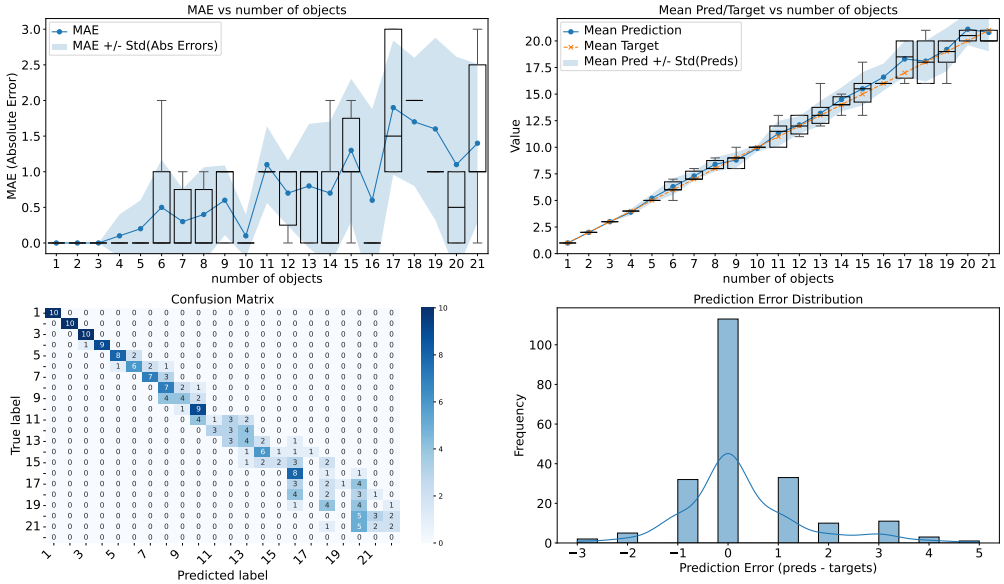


Figure 4: Results of GPT-4.1 on the counting task for the Chess dataset. Mean and std over 10 samples per level. More plots for this experiment can be found in Appendix B.2.

For each task, we obtain a detailed profile of the performance of each VLM (see Appendix B). For instance, we note that counting cards is more difficult than counting chess pieces. This might be due to the split of cards over the river and players, but both models struggle more with counting Poker cards. Surprisingly, GPT-4.1 almost always predicts a number of cards below the ground truth (by more than 1 element in average from 10 to 15 cards).

Through the OpenAI API, this experiment over 210 images used 210,630 input tokens (1003 tokens in average) and 2713 output tokens (13 tokens in average) for a total cost of 0.44$ and a total time of 734 seconds (3.5s in average).

4.2 Comparative benchmarks

Beyond individual diagnostics, we conduct a comparative evaluation of the following SoTA VLMs: GPT-4.1 [18] and GPT-4.1-mini [18] (via the OpenAI API), LLaMA-4-Scout (109B, 17B active) [19] and LLaMA-4-Maverick (400B, 17B active) [19] (via the Groq API https://groq.com/), Mistral 3.1 (24B) [53], Gemma3 (4B, 12B) [54], and LLaMA3.2-Vision [55, 56] (via Ollama https://ollama.com/ on a local machine with 2 NVIDIA GPUs RTX 5000 ADA and one NVIDIA GPU RTX 6000 ADA). All models are evaluated on the tasks listed in Table 1, using both chess and poker datasets with variations in object count, type, camera distance, blur, and card overlap. Averaged chess results are reported in Tables 2, 3 and Figure 5. Poker and full results appear in Appendix C.

In details, for the counting task (Table 3 and `Count` in Figure 5), the number of objects ranges from 1 to 21, or 1 to 4 when adding blur with 5 levels of blur (`Count Blur` in Figure 5). Regarding localization, we ask the VLM to detect the position of a single piece on a grid of size 4x4 (`Loc 4x4` in Figure 5) or the distance in terms of rows and columns of two pieces on 4x4 and 8x8 boards (`Loc 4x4 (2)` and `Loc 8x8 (2)` in Figure 5). Finally, the VLM has to identify a single piece or card for the identification task (`Identification` in Figure 5). We also add a variation of the camera distance from 1.7 to 7.5 meters for identification (`Identification Distance` in Figure 5).

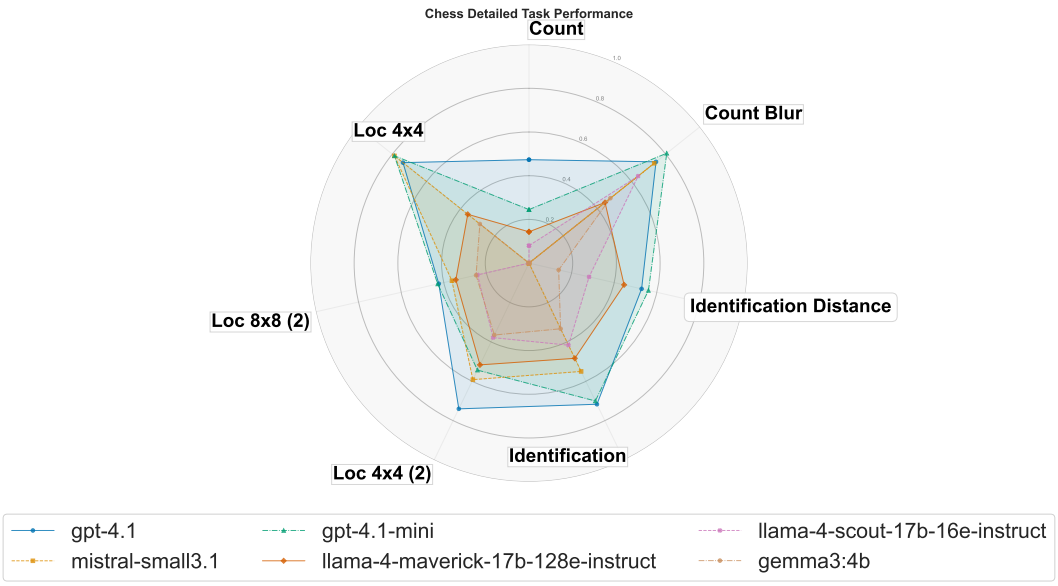


Figure 5: Model performance on chess tasks. Averaged accuracy across all samples; sample sizes are indicated per task: `Loc 8x8 (2)` (n=50), `Loc 4x4` (n=50), `Loc 4x4 (2)` (n=50), `Count` (n=850), `Count Blur` (n=191), `Identification Distance` (n=140), and `Identification` (n=50).

Overall, confirming previous works [2, 17, 16, 3], SoTA VLMs still face challenges to solve very basic visual tasks that are trivial to humans (e.g., counting pieces on a chessboard) (Figure 5 and Tables 2,3). Even high-performing models degrade under compositional task load or perceptual distortion. For instance, Llama-4-Scout exhibits sharp drops in accuracy on high-density chess boards (down to 29% at > 9 pieces, see Table 3) and struggles with card overlap in poker images. These findings suggest that current VLMs may rely on superficial visual heuristics rather than robust object-level understanding.

Results across all tasks are summarized in Tables 2 3 and Figure 5. GPT-4.1 and GPT-4.1-mini consistently outperform other models across tasks, GPT-4.1 showing slightly higher robustness in localization performances (MAE of 0.210 compared to GPT-4.1-mini's 0.480, Table 2). Among open-sourced models, Mistral-3.1 [53] and Llama-4-Scout show stronger performance on counting

tasks, but still lag in localization (Table 2) and identification under distance variation (Table 2). Notably, performance disparities are amplified under controlled stress conditions such as blur or increasing object count (Table 3).

Table 2: Average performance of VLMs on the Chess dataset. Mean and standard deviation over 50 images for each task. Localization is performed on 4x4 chessboards containing a single piece.

Model	Localization					Identification			
	Acc ↑	F1 ↑	MAE ↓	MSE ↓	NMAE ↓	Distance Variation		No Variation	
						Acc ↑	F1 ↑	Acc ↑	F1 ↑
gpt-4.1	0.83 (0.33)	0.91 (0.12)	0.17 (0.33)	0.17 (0.33)	0.13 (0.23)	**0.56** (0.34)	**0.72** (0.37)	0.68 (0.40)	0.81 (0.39)
gpt-4.1-mini	**0.98** (0.09)	**0.99** (0.06)	**0.02** (0.1)	**0.02** (0.09)	**0.01** (0.05)	0.59 (0.36)	0.74 (0.35)	**0.70** (0.33)	**0.82** (0.30)
llama-4-scout	0.29 (0.34)	0.45 (0.36)	2.63 (1.54)	10.4 (8.16)	1.45 (1.09)	0.45 (0.32)	0.62 (0.35)	0.43 (0.36)	0.61 (0.38)
llama-4-maverick	0.52 (0.40)	0.68 (0.34)	0.77 (0.70)	1.47 (1.68)	0.55 (0.48)	0.47 (0.34)	0.62 (0.36)	0.48 (0.39)	0.65 (0.40)
gemma3	0.42 (0.23)	0.60 (0.33)	1.93 (0.96)	5.37 (4.71)	0.94 (0.51)	0.14 (0.23)	0.24 (0.30)	0.35 (0.32)	0.52 (0.37)
gemma3:12b	0.48 (0.27)	0.49 (0.22)	0.97 (0.46)	1.60 (1.22)	0.61 (0.40)	0.48 (0.37)	0.65 (0.39)	0.45 (0.35)	0.62 (0.37)
llama3.2-vision	0.30 (0.28)	0.46 (0.33)	1.10 (0.51)	2.10 (1.56)	0.69 (0.46)	0.50 (0.32)	0.67 (0.33)	0.48 (0.38)	0.65 (0.39)
mistral-3.1	0.82 (0.30)	0.90 (0.22)	0.18 (0.30)	0.18 (0.30)	0.15 (0.25)	0.44 (0.37)	0.61 (0.39)	0.57 (0.38)	0.72 (0.38)

Table 3: Accuracy by object count range on the Chess dataset (Declarative instruction and Helpful preprompt). Mean and standard deviation across aggregations of levels, with 40 samples per level.

Model	Low (1–4 pcs)	Medium (5–9 pcs)	High (10-21 pcs)
GPT-4.1	0.94 (0.24)	0.75 (0.50)	0.54 (0.05)
GPT-4.1-mini	0.87 (0.33)	0.58 (0.46)	0.40 (0.23)
LLaMA-4-scout	0.75 (0.43)	0.61 (0.41)	0.27 (0.11)
LLaMA-4-Maverick	0.81 (0.39)	0.64 (0.34)	0.32 (0.21)
Gemma3	0.50 (0.50)	0.12 (0.26)	0.07 (0.34)
Gemma3:12b	0.63 (0.48)	0.34 (0.11)	0.17 (0.21)
LLaMA3.2-Vision	0.65 (0.43)	0.51 (0.17)	0.24 (0.32)
Mistral-small3.1	0.75 (0.46)	0.57 (0.18)	0.25 (0.19)

The radar chart (Figure 5) reveals fine-grained skill profiles: for instance, LLaMA-4-Scout performs relatively well on basic counting but drops sharply on 8×8 localization tasks, indicating poor spatial generalization. GPT-4.1 variants consistently outperform open-source models, especially in localization. While Gemma3-12B achieves high consistency in identification, it underperforms drastically in compositional and spatially complex setups, suggesting a lack of robust visual grounding.

To further understand the interplay between linguistic prompt design and visual task performance, we systematically evaluate combinations of preprompt strategies (Helpful, CoT, Neutral) and instruction types (Declarative, Missing Word) across all tasks. As shown in Figure 6, declarative instructions yield improvements over close-style prompts across all models (up to $+5\%$ accuracy on Poker counting for GPT-4.1-mini when combined with a Helpful preprompt, see Appendix C).

Finally, we examine the correlation using a sample of 80 real images. We recreate 80 Chess scenes from our chess count dataset, with the number of pieces varying from 1 to 8 (10 samples per image, as detailed in Appendix E). For each combination of model, preprompt, and instruction, we conduct our diagnostic analysis. After averaging predictions for each level over the 10 samples, preprompts, and instructions, we find Spearman and Pearson correlation scores exceeding 0.99 for all models, except for Gemma3-4B (0.93 and 0.91) and Gemma3-12B (0.64 and 0.75).

5 Discussion and future work

We introduce both a new methodology and a framework to precisely diagnose the perception abilities of VLMs. Our approach leverages procedurally generated synthetic data to isolate specific visual tasks and measure model performance across controlled difficulty levels. By disentangling low-level perceptual skills from high-order reasoning, our approach sheds light on VLM failure modes that are otherwise obscured in traditional benchmarks.

Discussion We believe that when selecting a vision-language model, the primary concern is how reliably it will handle the concrete tasks it will encounter in production—counting items on a conveyor

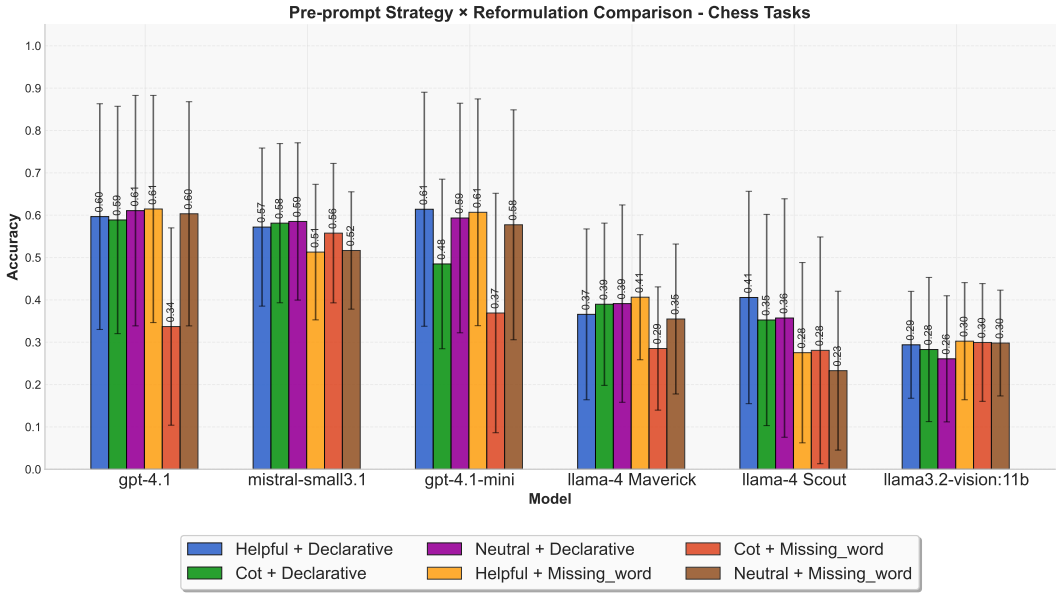


Figure 6: Impact of the choice of preprompt and instructions on accuracy (mean over all chess tasks, see Appendix C for a comparison per task).

belt, locating cracks, or interpreting color codes on labels. Therefore, although high-level evaluations are very useful to measure the progress of VLMs and get a rough idea of their abilities, a decoupled and interpretable assessment of their abilities is crucial to gain insights about what they can or cannot do in real-world scenarios [57, 58]. This implies the decomposition of high-level instruction following or image understanding into low-level functions. Focusing on the perception skills, a further decomposition could include measuring, brightness and color identification, etc. In this context, synthetic data are very useful as images and annotations of specific use cases are too costly to obtain.

Limitations and future work Our method and framework could benefit from several improvements. First of all, a better formalism of the evaluation process could lead to a smoother pipeline task-to-test→configuration→image and question generation→metrics. We plan to improve the abstraction classes to offer an easier integration of additional use cases (visual parameters, scene content, questions, etc). A more direct improvement would be to generate more realistic images, fitting real industrial scenarios, using tools from [59, 60]. Finally, for LLaVA-like architectures [20] with open-sourced parameters, identifying precisely the failing module (visual, alignment or textual module) could be done by training linear probes on top of the visual or merged representations as in [3].

References

[1] T. Bai, H. Liang, B. Wan, L. Yang, B. Li, Y. Wang, B. Cui, C. He, B. Yuan, and W. Zhang, "A survey of multimodal large language model from a data-centric perspective," *arXiv:2405.16640*, 2024.

[2] D. Campbell, S. Rane, T. Giallanza, C. N. De Sabbata, K. Ghods, A. Joshi, A. Ku, S. Frankland, T. Griffiths, J. D. Cohen, *et al.*, "Understanding the limits of vision language models through the lens of the binding problem," *Advances in Neural Information Processing Systems*, vol. 37, pp. 113436–113460, 2024.

[3] P. Rahmanzadehgervi, L. Bolton, M. R. Taesiri, and A. T. Nguyen, "Vision language models are blind," in *Proceedings of the Asian Conference on Computer Vision*, pp. 18–34, 2024.

[4] T. Rädsch, L. Mayer, S. Pavicic, A. E. Kavur, M. Knopp, B. Öztürk, K. Maier-Hein, P. F. Jaeger, F. Isensee, A. Reinke, *et al.*, "Bridging vision language model (vlm) evaluation gaps with a framework for scalable and cost-effective benchmark generation," *arXiv preprint arXiv:2502.15563*, 2025.

[5] H. Duan, J. Yang, Y. Qiao, X. Fang, L. Chen, Y. Liu, X. Dong, Y. Zang, P. Zhang, J. Wang, *et al.*, "Vlmevalkit: An open-source toolkit for evaluating large multi-modality models," in *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 11198–11201, 2024.

[6] H. Al-Tahan, Q. Garrido, R. Balestriero, D. Bouchacourt, C. Hazirbas, and M. Ibrahim, "Unibench: Visual reasoning requires rethinking vision-language beyond scaling," *arXiv preprint arXiv:2408.04810*, 2024.

[7] C. Fu, Y.-F. Zhang, S. Yin, B. Li, X. Fang, S. Zhao, H. Duan, X. Sun, Z. Liu, L. Wang, *et al.*, "Mme-survey: A comprehensive survey on evaluation of multimodal llms," *arXiv preprint arXiv:2411.15296*, 2024.

[8] Z. Li, X. Wu, H. Du, H. Nghiem, and G. Shi, "Benchmark evaluations, applications, and challenges of large vision language models: A survey," *arXiv preprint arXiv:2501.02189*, vol. 1, 2025.

[9] W.-L. Chiang, L. Zheng, Y. Sheng, A. N. Angelopoulos, T. Li, D. Li, H. Zhang, B. Zhu, M. Jordan, J. E. Gonzalez, and I. Stoica, "Chatbot arena: An open platform for evaluating llms by human preference," 2024.

[10] L. Chen, J. Li, X. Dong, P. Zhang, Y. Zang, Z. Chen, H. Duan, J. Wang, Y. Qiao, D. Lin, *et al.*, "Are we on the right way for evaluating large vision-language models?," *arXiv:2403.20330*, 2024.

[11] C. Schuhmann, R. Beaumont, R. Vencu, *et al.*, "Laion-5b: An open large-scale dataset for training next generation image-text models," in *NeurIPS Datasets and Benchmarks Track*, 2022.

[12] M. Cherti, C. Schuhmann, J. Bax, *et al.*, "Reproducible scaling laws for contrastive language-image learning," in *CVPR*, 2023.

[13] S. Ott, L. Kirsch, C. Kirsch, and Y. Bengio, "Mapping global dynamics of benchmark creation and saturation in artificial intelligence," *Nature Communications*, vol. 13, no. 1, p. 6724, 2022.

[14] C.-Y. Hsieh, H. Kervadec, and Z. Akata, "Sugarcrepe: Fixing hackable benchmarks for vision-language compositionality," in *NeurIPS Datasets and Benchmarks Track*, 2023.

[15] B. O. Community, "Blender - a 3d modelling and rendering package," 2018.

[16] J. Wang, Y. Ming, Z. Shi, V. Vineet, X. Wang, S. Li, and N. Joshi, "Is a picture worth a thousand words? delving into spatial reasoning for vision language models," *Advances in Neural Information Processing Systems*, vol. 37, pp. 75392–75421, 2024.

[17] C. Zhang and S. Wang, "Good at captioning, bad at counting: Benchmarking gpt-4v on earth observation data," *arXiv preprint arXiv:2401.17600*, 2024.

[18] A. Kumar, J. Yu, J. Hallman, M. Pokrass, A. Goucher, A. Ganesh, B. Cheng, B. McKinzie, B. Zhang, C. Koch, C. Wei, D. Medina, E. Wong, E. Kavanaugh, F. Bekerman, H. Hu, H. Ren, I. Singal, J. Kiros, J. Ai, J. Lin, J. Chien, J. McGrath, J. Lee, J. Wang, K. Lu, K. Georgiev, K. Luther, L. Jing, M. Schwarzer, M. Castro, N. Keskar, R. G. Lopes, S. Zhao, S. Chen, S. Sanjeev, T. Gordon, T. Sanders, W. Zhou, Y. Song, Y. Xie, Y. Jin, Z. Zhang, A. Ramesh, A. Low, A. Nichol, A. Gheorghe, A. Tulloch, B. Ghorbani, B. Minaiev, B. Houghton, C. Cole, C. Lu, E. Wong, H. Sheahan, J. Huh, J. Qin, J. Wang, J. Ward, J. Mo, J. Ruffell, K. Chen, K. Singhal, K. Nguyen, K. Hata, K. Liu, M. Trębacz, M. Lim, M. Pavlov, M. Chen, M. Griffiths, N. McAleese, N. Stathas, R. Samuel, R. T. Mullapudi, R. Zellers, S. Hu, S. Bi, S. Papay, S.-c. Yu, Y. Patil, Y. Zhang, A. Walker, A. Kamali, A. Wan, A. Wang, A. Singh, B. Leimberger, B. Hoover, B. Yu, C. Jatt, C. Ding, C. Chang, D. Kappler, D. Li, F. P. Such, J. Vembunarayanan, J. Florencio, K. King, L. Lv, L. Yang, L. Li, M. Liodakis, M. Hudnall, N. Handa, O. Godement, R. Madej, S. Chang, S. Fitzgerald, S. Wu, S. Fu, S. Hsieh, T. Sottiaux, Y. Dai, and Y. Liu, "Gpt-4.1, technical report," 2025. Research core contributors: Ananya Kumar, Jiahui Yu, John Hallman, Michelle Pokrass. Research contributors listed separately. Applied and scaling contributors listed separately.

[19] Meta, "The llama 4 herd." `https://ai.meta.com/blog/llama-4-multimodal-intelligence/`, 2025. Accessed May 2025.

[20] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *Advances in neural information processing systems*, vol. 36, pp. 34892–34916, 2023.

[21] X. Yue, Y. Ni, K. Zhang, T. Zheng, R. Liu, G. Zhang, S. Stevens, D. Jiang, W. Ren, Y. Sun, *et al.*, "Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi," in *CVPR*, 2024.

[22] Y. Liu, H. Duan, Y. Zhang, B. Li, S. Zhang, W. Zhao, Y. Yuan, J. Wang, C. He, Z. Liu, *et al.*, "Mmbench: Is your multi-modal model an all-around player?," *arXiv:2307.06281*, 2023.

[23] A. Singh, V. Natarjan, M. Shah, Y. Jiang, X. Chen, D. Batra, D. Parikh, and M. Rohrbach, "Towards vqa models that can read," in *CVPR*, 2019.

[24] B. Li, R. Wang, G. Wang, Y. Ge, Y. Ge, and Y. Shan, "Seed-bench: Benchmarking multimodal llms with generative comprehension," *arXiv:2307.16125*, 2023.

[25] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh, "Making the v in vqa matter: Elevating the role of image understanding in visual question answering," in *CVPR*, 2017.

[26] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[27] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[28] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, *et al.*, "Qwen2. 5 technical report," *arXiv preprint arXiv:2412.15115*, 2024.

[29] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[30] M. Cao, P. Hu, Y. Wang, J. Gu, H. Tang, H. Zhao, J. Dong, W. Yu, G. Zhang, I. Reid, *et al.*, "Video simpleqa: Towards factuality evaluation in large video language models," *arXiv preprint arXiv:2503.18923*, 2025.

[31] W. Peng, L. Meng, Y. Chen, Y. Xie, Y. Liu, T. Gui, H. Xu, X. Qiu, Z. Wu, and Y.-G. Jiang, "Inst-it: Boosting multimodal instance understanding via explicit visual prompt instruction tuning," *arXiv preprint arXiv:2412.03565*, 2024.

[32] M. F. Qharabagh, M. Ghofrani, and K. Fountoulakis, "Lvlm-count: Enhancing the counting ability of large vision-language models," *arXiv preprint arXiv:2412.00686*, 2024.

[33] N. Amini-Naieni, T. Han, and A. Zisserman, "Countgd: Multi-modal open-world counting," *Advances in Neural Information Processing Systems*, vol. 37, pp. 48810–48837, 2024.

[34] Y. Jeon, S. Lee, J. Kim, and J.-P. Heo, "Mutually-aware feature learning for few-shot object counting," *Pattern Recognition*, vol. 161, p. 111276, 2025.

[35] Z. Huang, M. Dai, Y. Zhang, J. Zhang, and H. Shan, "Point segment and count: A generalized framework for object counting," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17067–17076, 2024.

[36] I. Stogiannidis, S. McDonagh, and S. A. Tsaftaris, "Mind the gap: Benchmarking spatial reasoning in vision-language models," *arXiv preprint arXiv:2503.19707*, 2025.

[37] M. Acharya, K. Kafle, and C. Kanan, "Tallyqa: Answering complex counting questions," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 8076–8084, 2019.

[38] P. Chattopadhyay, R. Vedantam, R. R. Selvaraju, D. Batra, and D. Parikh, "Counting everyday objects in everyday scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1135–1144, 2017.

[39] Q. Wang, J. Gao, W. Lin, and Y. Yuan, "Pixel-wise crowd understanding via synthetic data," *International Journal of Computer Vision*, vol. 129, no. 1, pp. 225–245, 2021.

[40] J. Gao, L. Zhao, and X. Li, "Nwpu-moc: a benchmark for fine-grained multicategory object counting in aerial images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 62, pp. 1–14, 2024.

[41] F. Shiri, X.-Y. Guo, M. G. Far, X. Yu, G. Haffari, and Y.-F. Li, "An empirical analysis on spatial reasoning capabilities of large multimodal models," *arXiv preprint arXiv:2411.06048*, 2024.

[42] A.-C. Cheng, H. Yin, Y. Fu, Q. Guo, R. Yang, J. Kautz, X. Wang, and S. Liu, "Spatialrgpt: Grounded spatial reasoning in vision language models," *arXiv preprint arXiv:2406.01584*, 2024.

[43] H. Bao, Y. Huang, Y. Wang, J. Ye, X. Wang, X. Chen, Y. Zhao, T. Zhou, M. Elhoseiny, and X. Zhang, "Autobench-v: Can large vision-language models benchmark themselves?," *arXiv preprint arXiv:2410.21259*, 2024.

[44] M. Żelaszczyk and J. Mańdziuk, "Text-to-image cross-modal generation: A systematic review," *arXiv preprint arXiv:2401.11631*, 2024.

[45] H. Liu, W. Xue, Y. Chen, D. Chen, X. Zhao, K. Wang, L. Hou, R. Li, and W. Peng, "A survey on hallucination in large vision-language models," *arXiv preprint arXiv:2402.00253*, 2024.

[46] J. Zhang, W. Huang, Z. Ma, O. Michel, D. He, T. Gupta, W.-C. Ma, A. Farhadi, A. Kembhavi, and R. Krishna, "Task me anything," *arXiv preprint arXiv:2406.11775*, 2024.

[47] J. Zhang, L. Xue, L. Song, J. Wang, W. Huang, M. Shu, A. Yan, Z. Ma, J. C. Niebles, S. Savarese, *et al.*, "Provision: Programmatically scaling vision-centric instruction data for multimodal language models," *arXiv preprint arXiv:2412.07012*, 2024.

[48] H. Tang and K. Jia, "A new benchmark: On the utility of synthetic data with blender for bare supervised learning and downstream domain adaptation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15954–15964, 2023.

[49] J. H. Cho, A. Madotto, E. Mavroudi, T. Afouras, T. Nagarajan, M. Maaz, Y. Song, T. Ma, S. Hu, S. Jain, *et al.*, "Perceptionlm: Open-access data and models for detailed visual understanding," *arXiv preprint arXiv:2504.13180*, 2025.

[50] A. Mumuni, F. Mumuni, and N. K. Gerrar, "A survey of synthetic data augmentation methods in machine vision," *Machine Intelligence Research*, vol. 21, no. 5, pp. 831–869, 2024.

[51] W. Chow, J. Mao, B. Li, D. Seita, V. Guizilini, and Y. Wang, "Physbench: Benchmarking and enhancing vision-language models for physical world understanding," *arXiv preprint arXiv:2501.16411*, 2025.

[52] D. Paglieri, B. Cupiał, S. Coward, U. Piterbarg, M. Wolczyk, A. Khan, E. Pignatelli, Ł. Kuciński, L. Pinto, R. Fergus, *et al.*, "Balrog: Benchmarking agentic llm and vlm reasoning on games," *arXiv preprint arXiv:2411.13543*, 2024.

[53] M. AI, "Mistral small 3.1." `https://mistral.ai/news/mistral-small-3-1`, 2025. Accessed May 2025.

[54] G. Team, A. Kamath, J. Ferret, S. Pathak, N. Vieillard, R. Merhej, S. Perrin, T. Matejovicova, A. Ramé, M. Rivière, *et al.*, "Gemma 3 technical report," *arXiv preprint arXiv:2503.19786*, 2025.

[55] Meta, "Llama 3.2," 2025. Accessed May 2025.

[56] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[57] S. Ott, A. Barbosa-Silva, K. Blagec, J. Brauner, and M. Samwald, "Mapping global dynamics of benchmark creation and saturation in artificial intelligence," *Nature Communications*, vol. 13, no. 1, p. 6793, 2022.

[58] A. Reuel, A. Hardy, C. Smith, M. Lamparth, M. Hardy, and M. J. Kochenderfer, "Betterbench: Assessing ai benchmarks, uncovering issues, and establishing best practices," *arXiv preprint arXiv:2411.12990*, 2024.

[59] M. Roberts, J. Ramapuram, A. Ranjan, A. Kumar, M. A. Bautista, N. Paczan, R. Webb, and J. M. Susskind, "Hypersim: A photorealistic synthetic dataset for holistic indoor scene understanding," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 10912–10922, 2021.

[60] M. Denninger, D. Winkelbauer, M. Sundermeyer, W. Boerdijk, M. W. Knauer, K. H. Strobl, M. Humt, and R. Triebel, "Blenderproc2: A procedural pipeline for photorealistic rendering," *Journal of Open Source Software*, vol. 8, no. 82, p. 4901, 2023.

[61] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24824–24837, 2022.

A Framework overview

In this section, we give an overview of our framework in practice and we refer the reader to Appendix D for details.

A.1 Overview of the Dataset Construction Process

This subsection provides an overview of our synthetic dataset generation pipeline, from task specification to annotated image production. We illustrate the main stages through a chess example, and highlight the underlying structure and key parameters. Readers seeking a complete technical specification may refer to Appendix D.

Step 1: Task Definition and Parameter Space Design. The process begins by identifying the core task, selecting the relevant variables to control and their range or value. For example, for a chess piece counting task, variables might include the number and type of pieces, board size, camera viewpoint, and lighting conditions. Each parameter is formally encoded in configuration schemas, typically as Python dataclasses or structured YAML/JSON dictionaries.

Step 2: Scene Configuration via Strongly-Typed Models. All scene components (e.g., board, pieces, camera, lighting, materials) are described by hierarchical configuration models. For instance, a `CameraModel` encodes camera distance, elevation angle, and randomization settings, while a `PieceModel` specifies piece type, position, color, and size. This design enables programmatic control, type safety, and easy extensibility. An illustrative Python configuration snippet for a simple chess scene:

```
scene_config = {
    "board": {
        "rows": 8,
        "columns": 8,
        "board_material": {"color": (0.7, 0.6, 0.5, 1.0)}
    },
    "pieces": [
        {"type": "king", "position": (0, 4),
        "color": (0.9, 0.9, 0.9, 1.0)},
        {"type": "queen", "position": (7, 3),
        "color": (0.1, 0.1, 0.1, 1.0)}
    ],
    "camera": {"distance": "medium", "angle": 60.0},
    "lighting": {"lighting": "high"},
    "noise": {"blur": "low", "table_texture": "medium"}
}
```

Alternatively, the same setup can be specified in YAML for experiment management:

```
board:
  rows: 8
  columns: 8
  board_material:
    color: [0.7, 0.6, 0.5, 1.0]
pieces:
  - type: king
    position: [0, 4]
    color: [0.9, 0.9, 0.9, 1.0]
  - type: queen
    position: [7, 3]
    color: [0.1, 0.1, 0.1, 1.0]
camera:
  distance: medium
  angle: 60.0
lighting:
  lighting: high
noise:
  blur: low
  table_texture: medium
```

Step 3: 3D Scene Construction in Blender. At the core of our pipeline is Blender, an open-source 3D engine, controlled programmatically through its Python API (`bpy`). Using our configuration models, the pipeline builds each scene element:

- **Board and Pieces:** Geometry and material are either procedurally generated or loaded from pre-existing assets. Piece positioning, scale, and orientation are set via configuration.
- **Camera and Lighting:** Camera parameters (distance, angle, random jitter) and lighting setups (key, fill, back light) are instantiated from their model descriptions.
- **Materials and Noise:** Surface materials, textures, and noise effects (e.g., blur, table surface complexity) are applied based on parameter presets or custom values.

This approach provides full flexibility: as illustrated below (Figure 7), one can systematically vary the number of pieces on the chess board. It is possible to do the same with all other visual parameters introduced in the framework: camera distance, etc.

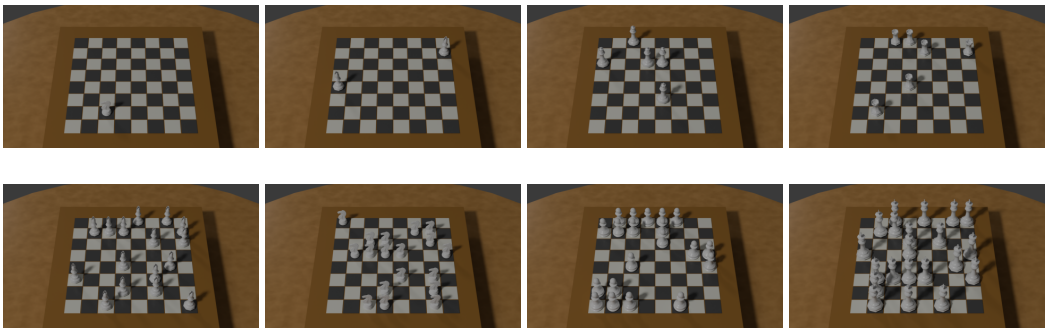


Figure 7: Examples of generated chess images with variation of the number of pieces on the board.

Step 4: Automated Rendering and Legend Generation. Once the scene is constructed, images are rendered at the specified resolution and file format. Crucially, alongside each image, the pipeline generates a *legend*—a comprehensive metadata file (text and JSON) capturing every scene parameter, object attribute, and applied transformation. This ensures reproducibility and enables automated extraction of ground-truth answers for downstream evaluation.

For example, the legend for a chess image includes: board geometry, piece types and positions, camera settings, noise levels, and any random seeds used. This metadata underpins both the traceability of the dataset and the automated benchmarking of model predictions.

```
{
  "board": {"rows": 8, "columns": 8},
  "pieces": [
    {"type": "king", "position": [0, 4],
    "color": [0.9, 0.9, 0.9, 1.0]},
    {"type": "queen", "position": [7, 3],
    "color": [0.1, 0.1, 0.1, 1.0]}
  ],
  "camera": {"distance": "medium", "angle": 60.0},
  "noise": {"blur": "low"}
}
```

Step 5: Question/Answer Generation and Extraction. For each rendered image, the system instantiates visual reasoning questions (e.g., "How many pieces are there in the image?") and uses the legend to automatically extract ground-truth answers. The question and answer generation logic is domain-agnostic, supporting both chess and poker (see Section D.2 for details).

Advanced Features and Extensibility.

- **Parameter Randomization:** The framework supports controlled randomization of variables (e.g., piece placement, lighting intensity), enabling statistical evaluation of model robustness.
- **Presets and Customization:** Named presets (e.g., `"high"` blur, `"low"` lighting) simplify experiment setup, while custom values allow fine-grained control.
- **Scalable Combinatorial Generation:** For large-scale experiments, variable combination generators exhaustively or randomly instantiate all possible scene configurations, ensuring coverage of the experimental space.
- **Domain Extension:** Although illustrated here for chess, the modular architecture extends naturally to other domains (e.g., poker, go), supporting new object types, rules, and scene structures with minimal changes.

End-to-End Example. A minimal working example of the pipeline might involve:

1. Defining a YAML file specifying a range of piece counts for chess images.
2. Running the dataset generator, which builds each scene in Blender, applies the specified configurations, and renders the images.
3. Saving, for each image, a legend file containing all parameters, and extracting answers for a suite of questions.
4. Using the resulting image-question-answer triplets for downstream model training or evaluation.

In summary, our pipeline provides a fully automated, extensible, and transparent process for generating annotated visual reasoning datasets, with precise experimental control over all scene elements. For full technical details, schema definitions, and advanced usage, please refer to Appendix D.

B Diagnostic - additional experiments

B.1 General observations

We present additional results from the diagnostic experiment on the task set introduced in Section 4. We do not systematically include all results for every task, but instead provide representative diagnostics for both GPT-4.1 and LLaMA-4-Scout.

Overall, the following key observations can be made:

- **Counting**: These diagnostics reveal model biases in specific domains. GPT-4.1 tends to slightly overestimate the count of large object numbers in the Chess dataset, while significantly underestimating the count of Poker cards. Notably, these biases appear to be model-dependent, as LLaMA-4-Scout demonstrates a minor tendency to underestimate object counts in the Chess dataset, in contrast to GPT-4.1.
- **Counting with blur**: Both GPT-4.1 and LLaMA-4-Scout exhibit an overestimation pattern in object counting as blur increases, which may be explained by the fact that higher levels of blur can merge pixels, creating the illusion of objects where none exist.
- **Localization of a single Chess piece on a 4x4 grid**: This experiment highlights the contextual bias exhibited by the LLaMA-4-Scout model, as it consistently predicts out-of-bounds rows and columns, likely inferring a standard 8x8 chessboard.
- **Localization of a single card on a 3x3 Poker grid**: Both models demonstrate a decline in performance when the element is positioned in the middle column of the poker scene. While LLaMA-4-Scout tends to overestimate column locations, making again out-of-bounds predictions, its performance remains more accurate for rows. This may be attributed to the rectangular shape of the table, leading to a grid where rows are more compact than columns, and highlights the importance of considering the geometry of the scene.
- **Relative localization of two Chess pieces on an 8x8 grid**: Both models' performance declines rapidly as the row distance between the two pieces increases, with a clear underestimation of their relative distance. These tendencies contrast with those observed when

locating a single chess piece on a 4x4 grid, emphasizing the importance of conducting both diagnostics.

- **Identification with camera distance (Chess)**: As anticipated, the performance appears to decline as the distance to the chess scene increases.
- **Counting with Horizontal Overlap (Poker cards)**: Both models tend to underestimate the number of cards with horizontal overlap. Additionally, LLaMA-4-Scout seems to particularly struggle with the concept of cards encroaching upon one another, as its underestimation steadily increases with the level of overlap.

B.2 Counting (Chess)

We present the diagnostic results for the *counting* task on the Chess dataset. The following question is asked with a debiased preprompt and a declarative instruction:

"This is not a real chess game. The number of each piece and their position can vary arbitrarily. Just focus on answering the following question based on the visual content. How many pieces are there in the image? Respond in a declarative format: 'The number of pieces in the image is:'"

Examples of images can be found below in Figure 8:

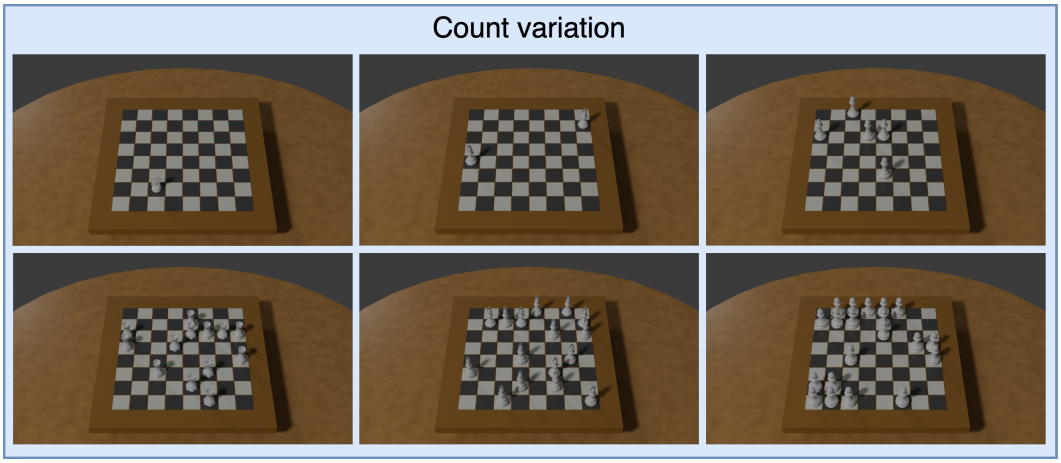


Figure 8: Variations of the number of Chess pieces (cropped images with overlap for display purposes)

Results

- GPT-4.1 (Figure 9). The model achieves 100% accuracy up to 3 objects, beyond which it begins to make errors. The Mean Absolute Error (MAE) remains near 1 up to 14 objects, after which it starts to increase. The prediction error appears to be approximately centered, with the model showing a slight tendency to overestimate the number of objects.
- LLaMA-4-Scout (Figure 10). The model also remains at 100% accuracy up to 3 chess pieces, beyond which it begins to make errors. However, the Mean Absolute Error (MAE) rapidly increases, reaching an average of 2, with outliers that significantly impact the average score. The distribution of prediction errors is approximately centered, with frequent slight underestimates of the object count and a few notable overestimates at higher levels.

While both models achieve perfect scores up to 3 objects, these diagnostics help to reveal a slight bias in their counting abilities. GPT-4.1 exhibits a tendency to slightly overestimate the number of objects, whereas LLaMA-4-Scout tends to slightly underestimate the count.

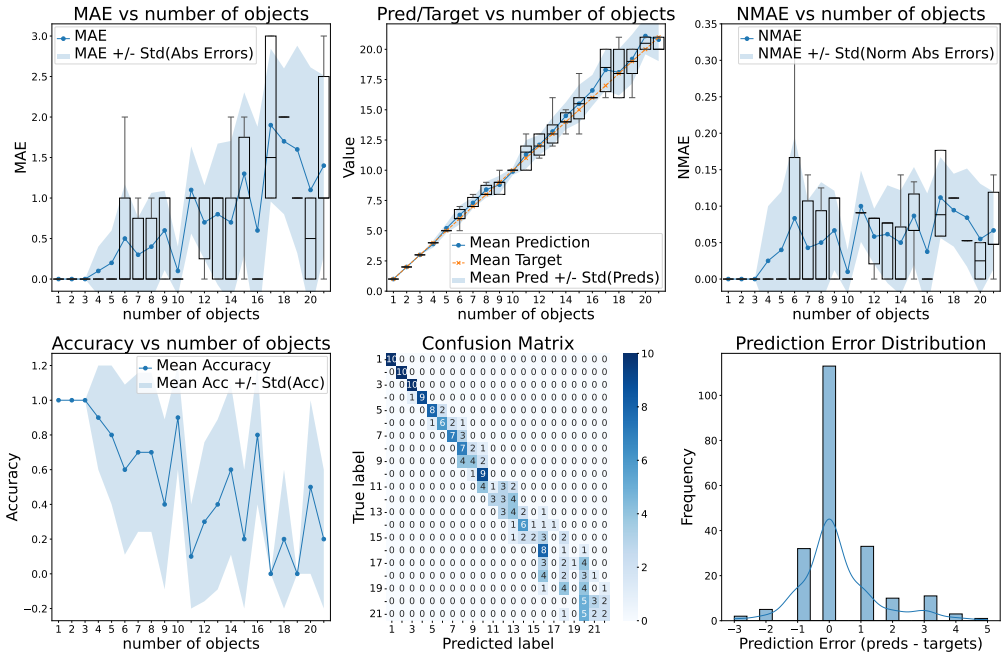


Figure 9: Results of GPT-4.1 for the counting question on the Chess dataset. Mean and standard deviation are computed over 10 samples per level.

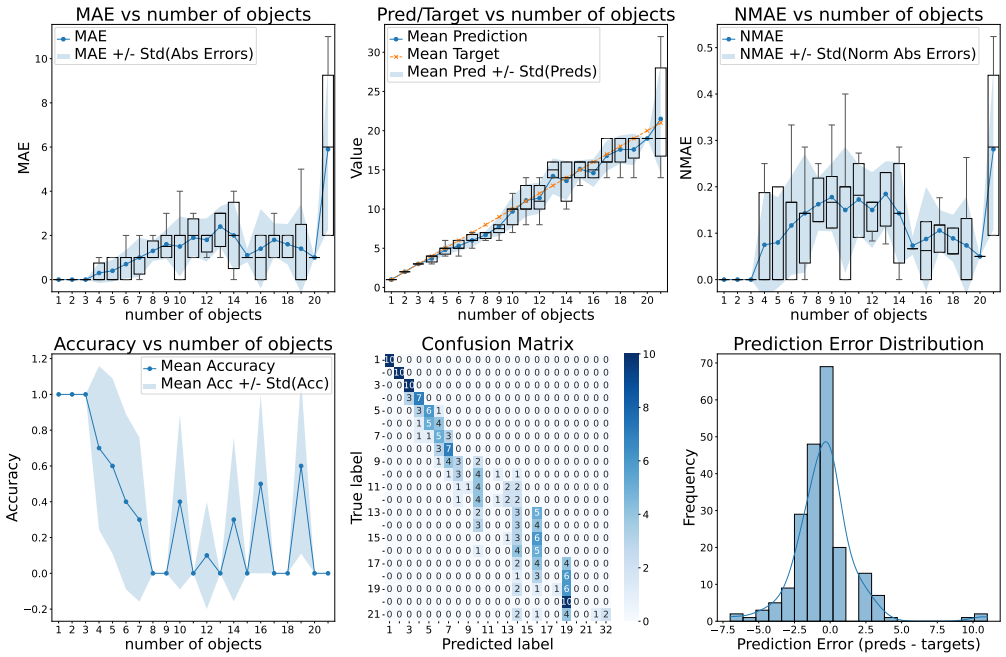


Figure 10: Results of LLaMA 4 for the counting question on the Chess dataset. Mean and standard deviation are computed over 10 samples per level.

B.3 Counting (Poker)

We present the diagnostic results for the *counting* task on the Poker dataset. The following question is asked with a debiased prepompt and a declarative instruction:

"This is not a real poker game. The cards and their position can vary arbitrarily. Just focus on answering the following question based on the visual content. How many cards are present in the entire scene (including all hands and community cards)? Respond in a declarative format: 'The number of cards in the image is:'"

Examples of images can be found below in Figure 11:

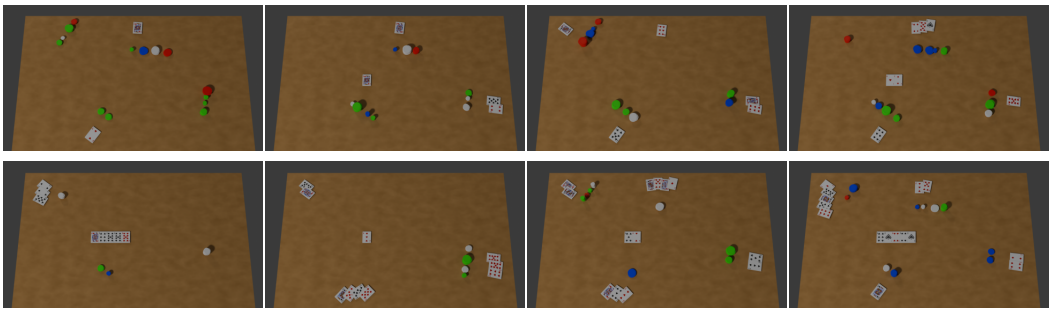


Figure 11: Variations of the number of cards in a Poker scenes (cropped images with overlap for display purposes)

Results GPT-4.1 (Figure 12). Across 15 poker scenes, the model makes only one or two errors when there are up to two cards on the table. With up to 9 cards, it accurately counts approximately half of the time (accuracy of around 50%), with an average Mean Absolute Error (MAE) close to 1. Beyond this point, the model consistently underestimates the number of cards by 2 or 3 on average, leading to a significant decline in accuracy. The counting task appears to be more challenging, which may be attributed to the fact that the poker game setup exhibits less regular geometries compared to the chess setup.

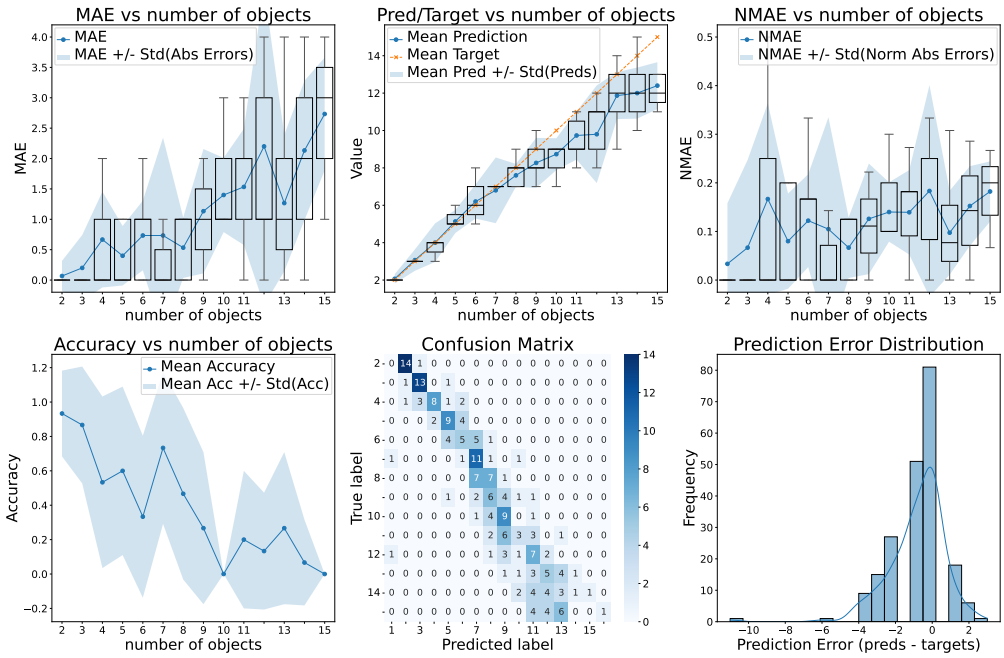


Figure 12: Results of GPT-4.1 for the counting question (*How many cards are present in the entire scene?*) on the Poker dataset. Mean and standard deviation over 15 samples per level.

B.4 Counting with blur (Chess)

We present the diagnostic results for the *counting with blur* task on the Chess dataset. The following question is asked with a debiased preprompt and a declarative instruction:

"This is not a real chess game. The number of each piece and their position can vary arbitrarily. Just focus on answering the following question based on the visual content. How many pieces are there in the image? Respond in a declarative format: 'The number of pieces in the image is:'"

Examples of images can be found below in Figure 13:

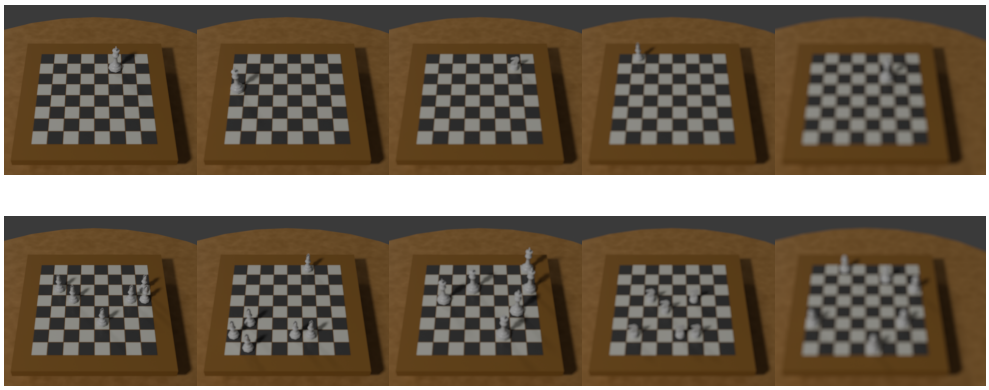
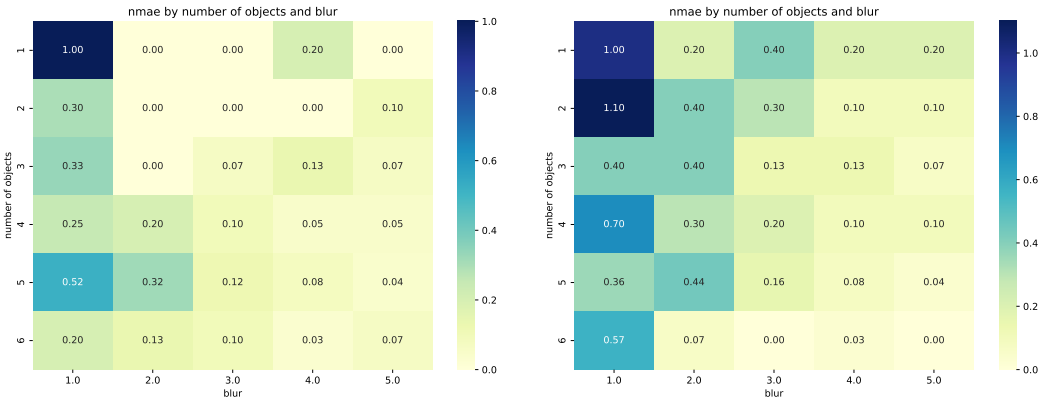


Figure 13: Variations of the five levels of blur for a chess scene, ranging from low (left) to high (right) levels. The images on the top display a single piece, while those on the bottom show multiple pieces.

Results

- GPT-4.1 (Figure 14). The model's performance clearly deteriorates as the level of blur increases, with a sharp decline in accuracy at the highest blur level (level 1). GPT-4.1 tends to overestimate the number of pieces as blur increases, which is expected, as higher levels of blur can merge pixels, creating the illusion of objects where none exist. Figure 16a illustrates that the primary source of difficulty is the increase in blur, rather than the rise in the number of pieces, as the Normalized Mean Absolute Error (NMAE) remains relatively stable (or even decreases for larger numbers) for a fixed blur value.

- LLaMA-4-Scout (Figure 15). LLaMA-4-Scout exhibits a similar overestimation pattern, with a marginally higher MAE and the distribution of prediction errors slightly shifted to the right. Figure 15 demonstrates that this model experiences greater difficulty with larger numbers of objects under high blur compared to GPT-4.1.



(a) GPT-4.1 (b) LLaMA-4-Scout

Figure 16: Normalized MAE of (a) GPT-4.1 and (b) LLaMA-4-Scout on cross-variations of blur and number of pieces. Higher values mean less blur. Average over 5 samples per cell.

B.5 Localization of a single piece on a 4x4 grid (Chess)

We present the diagnostic results for the *localization of a single piece* task on a 4x4 Chess board. The following questions are asked with a debiased preprompt and a declarative instruction:

"This is not a real chess game. The number and position of the pieces can vary arbitrarily. Just focus on answering the following question based on the visual content. Numbering the columns from left to right, starting with 0, on which column is the piece on the board? Respond in a declarative format."

"This is not a real chess game. The number and position of the pieces can vary arbitrarily. Just focus on answering the following question based on the visual content. Numbering the rows from top to bottom, starting with 0, on which row is the piece on the board? Respond in a declarative format."

Examples of images can be found below in Figure 17:

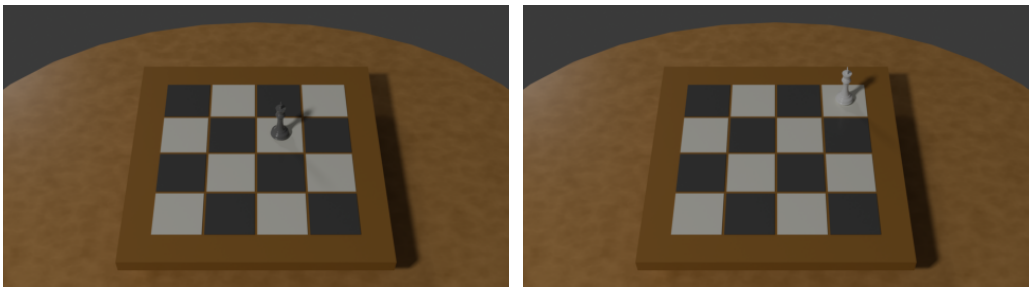


Figure 17: Illustration of the localization task for a single Chess piece on a 4x4 grid

Results

- GPT-4.1 (Figures 18, 19). The performance of the model is rather stable across the piece position, with an error of at most one column or row.
- LLaMA-4-Scout (Figure 20, 21). The model performs worse than GPT-4.1, exhibiting a clear overestimation bias for rows and columns beyond the first. In both cases, it frequently predicts row or column "7," likely reflecting a bias towards identifying an 8x8 chessboard and its extremities. This leads to a right-shifted distribution of prediction errors for both row and column localization.

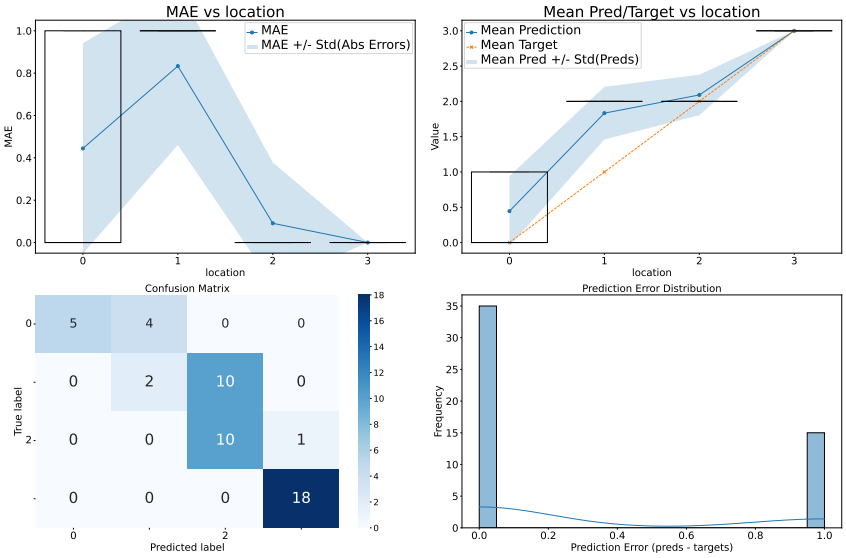


Figure 18: GPT-4.1 results for horizontal (column) localization of a single chess piece on a 4x4 grid. 50 samples overall.

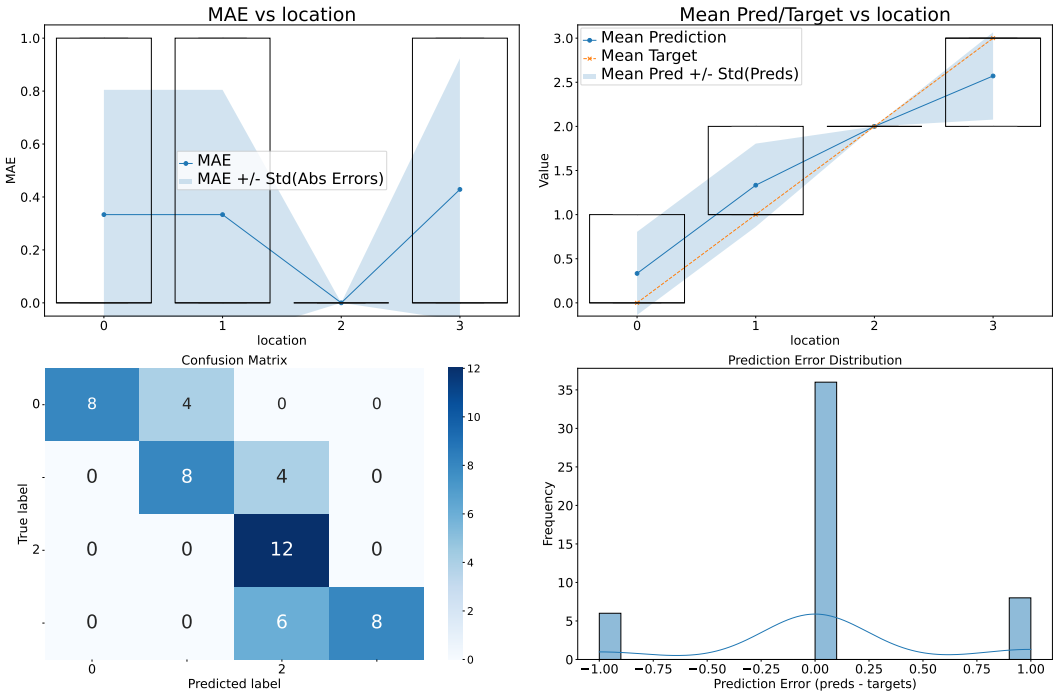


Figure 19: GPT-4.1 results for vertical (row) localization of a single chess piece on a 4x4 grid. 50 samples overall.

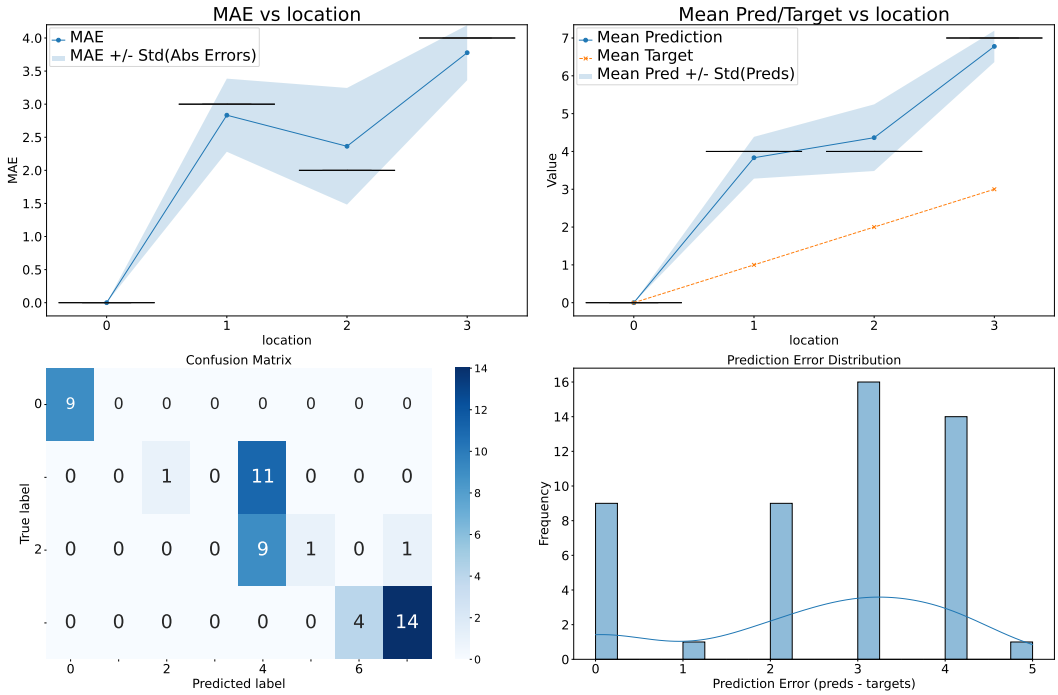


Figure 20: LLaMA-4-Scout results for horizontal (column) localization of a single chess piece on a 4x4 grid.

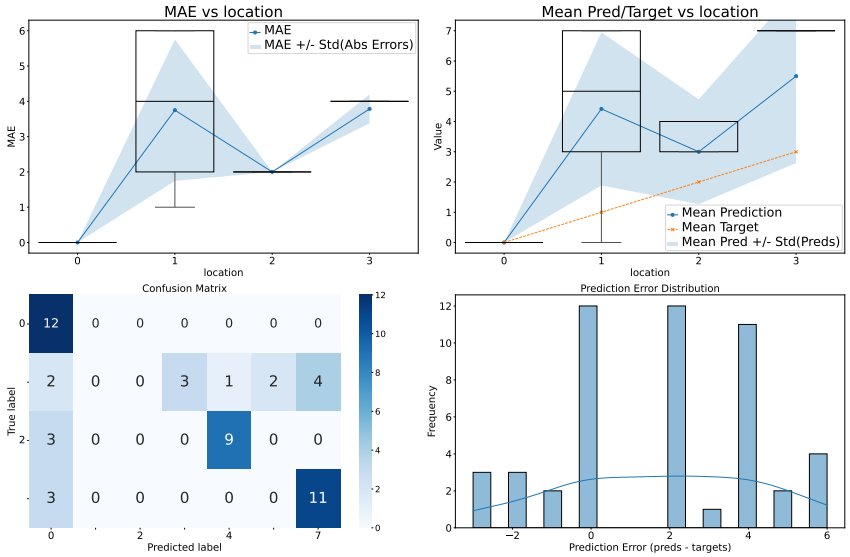


Figure 21: LLaMA-4-Scout results for vertical (row) localization of a single chess piece on a 4x4 grid. 50 samples overall.

B.6 Localization of a single card on a 3x3 grid (Poker)

We present the diagnostic results for the *localization of a single card* task on a 3x3 Poker grid. The following questions are asked with a debiased preprompt and a declarative instruction:

> "This is not a real poker game. The cards and their position can vary arbitrarily. Just focus on answering the following question based on the visual content. Numbering the rows from top to

bottom, starting with 0, on which row is the card? Respond in a declarative format: 'The row number of the grid where card X is located is:'"

"This is not a real poker game. The cards and their position can vary arbitrarily. Just focus on answering the following question based on the visual content. Numbering the columns from left to right, starting with 0, on which column is the card? Respond in a declarative format: 'The column number of the grid where card X is located is:'"

Examples of images can be found below in Figure 22:

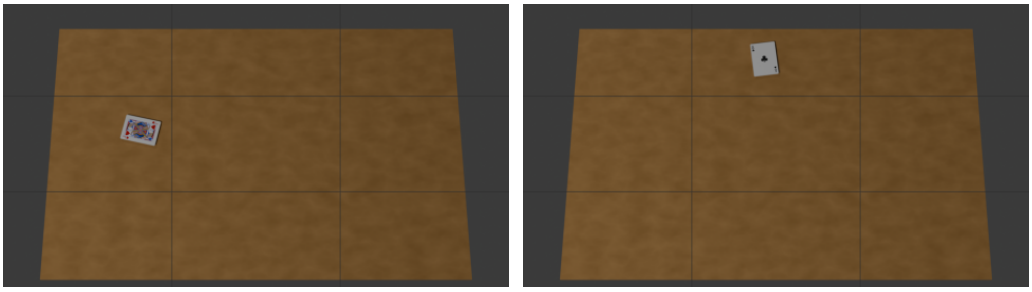


Figure 22: Illustration of the localization task of a single Poker card on a 3x3 grid

Results

- GPT-4.1 (Figures 23, 24). The model performs well at the edges, but its performance notably declines when the card is positioned in the middle row or column. However, the model maintains a centered distribution of prediction errors.

- LLaMA-4-Scout (Figure 25, 26). Similarly, LLaMA-4-Scout performs worse when the card is positioned in the middle column, although with a significantly higher MAE. There is also a clear overestimation of the column number when the card is centrally located, leading to a prediction error distribution shifted to the right, without a clearly normalized distribution. Notably, the model tends to identify more rows and columns than are present in the grid. In terms of row localization, performance remains stable across all row positions, with a centralized prediction error distribution and lower MAE. Notably, the model exhibits fewer instances of overestimating higher row numbers, which may be attributed to the rectangular geometry of the grid, where rows are more condensed than columns.

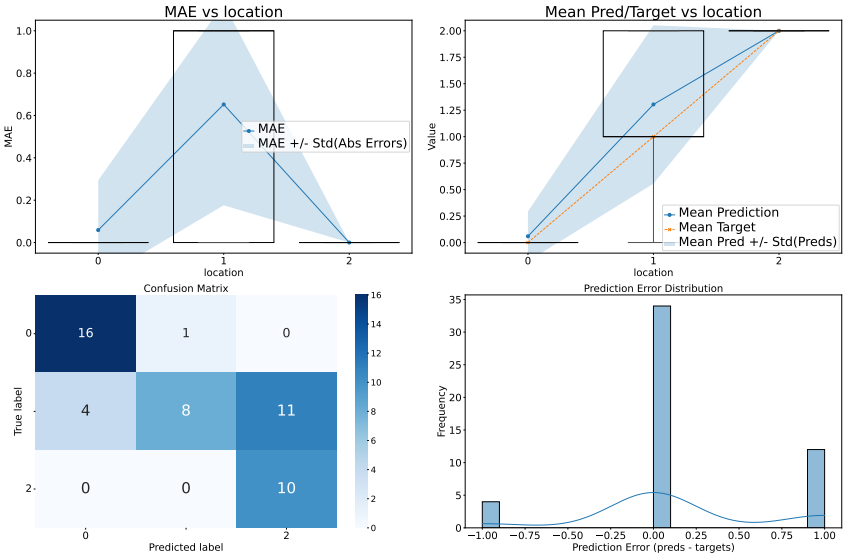


Figure 23: GPT-4.1 results for horizontal (column) localization of a single card on a 3x3 Poker grid. 50 samples overall.

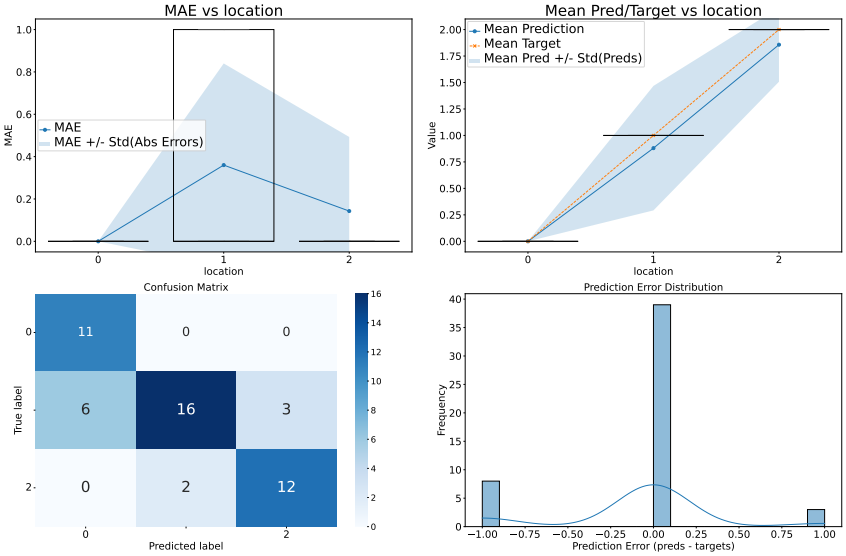


Figure 24: GPT-4.1 results for vertical (row) localization of a single card on a 3x3 Poker grid. 50 samples overall.

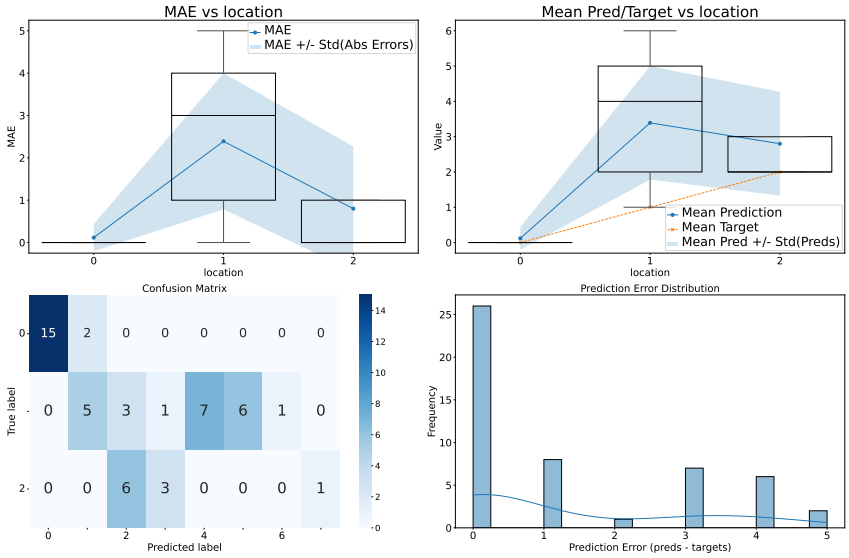


Figure 25: LLaMA results for horizontal (column) localization of a single card on a 3x3 Poker grid. 50 samples overall.

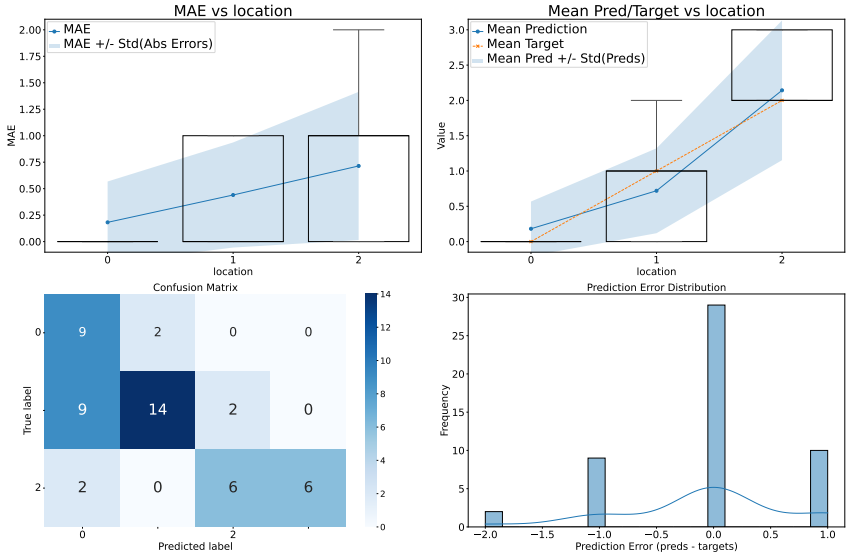


Figure 26: LLaMA results for vertical (row) localization of a single card on a 3x3 Poker grid. 50 samples overall.

B.7 Relative localization of two pieces on an 8x8 grid (Chess)

We present the diagnostic results for the *relative localization of two pieces* task on an 8x8 Chess grid. The following question is asked with a debiased preprompt and a declarative instruction:

"This is not a real chess game. The number and position of the pieces can vary arbitrarily. Just focus on answering the following question based on the visual content. Numbering the rows from top to bottom, starting with 0, on which row is the piece on the board? Respond in a declarative format."

Examples of images can be found below in Figure 27:

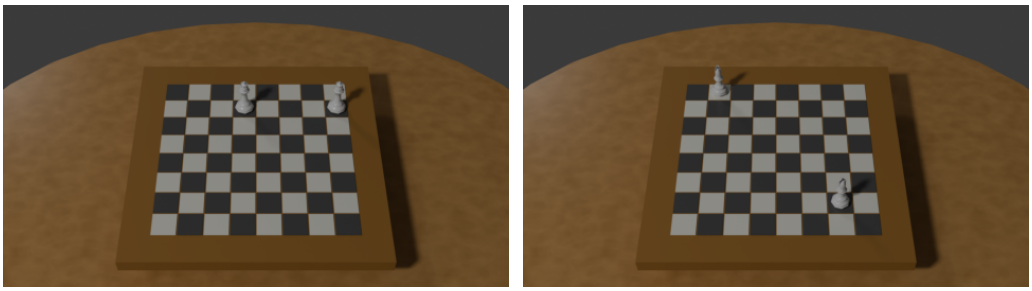


Figure 27: Illustration of the relative localization of two Chess pieces on an 8x8 grid

Results

- GPT-4.1 (Figure 28). The model accurately identifies when the two pieces are on the same row, but its performance declines rapidly as the distance between the pieces increases, reaching zero accuracy when the distance is 4 rows or greater. The distribution of prediction errors appear to be shifted to the left. GPT-4.1 demonstrates a clear underestimation of the distance when it exceeds 2 rows, with an MAE of approximately 1.5 for larger distances.
- LLaMA-4-Scout (Figure 29). Similarly, LLaMA-4-Scout's performance declines as the distance between the two pieces increases, with accuracy dropping to zero for distances greater than 4 rows. However, the underestimation bias is more pronounced, with an average prediction error of approximately 3 rows for distances of 6 or 7 rows. It consistently predicts a row distance of no more than 4. The distribution of prediction errors is distinctly shifted to the left, centered around -1.

While LLaMA-4-Scout exhibited an overestimation bias when localizing a single piece on a 4x4 grid (cf. Figure 21), it demonstrates a clear underestimation bias for the relative localization of two pieces on an 8x8 grid. This underscores the distinction between the two tasks and emphasizes the importance of conducting both diagnostics. In the first case, the model seems to infer knowledge of a standard 8x8 chessboard, while in the second, it consistently predicts a row distance of no more than 4, failing to account for the full size of the grid.

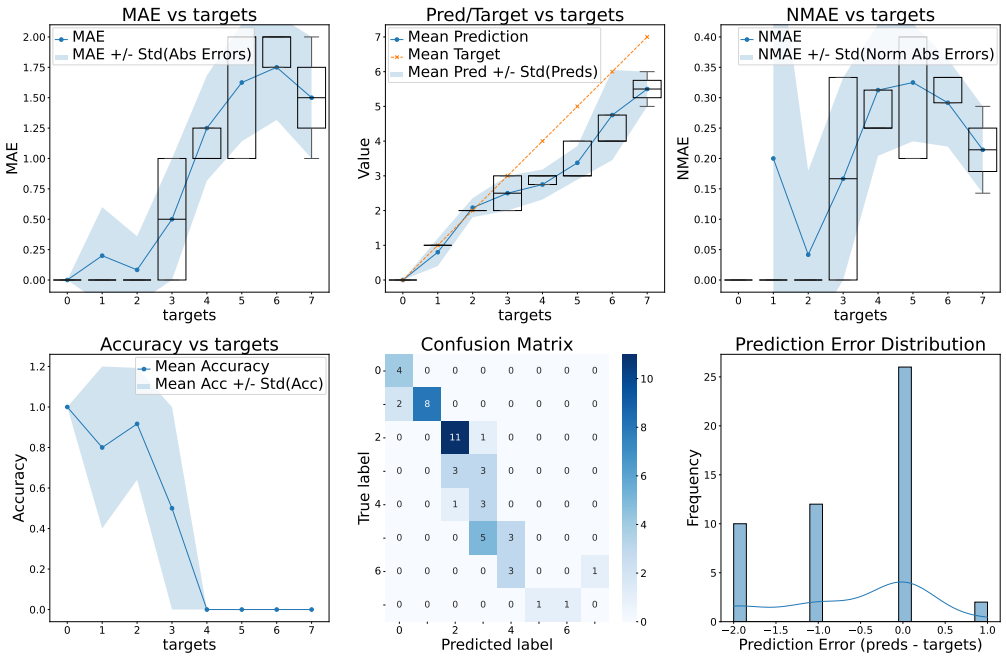


Figure 28: GPT-4.1 results for vertical distance between two pieces on an 8x8 Chess grid. 50 samples overall.

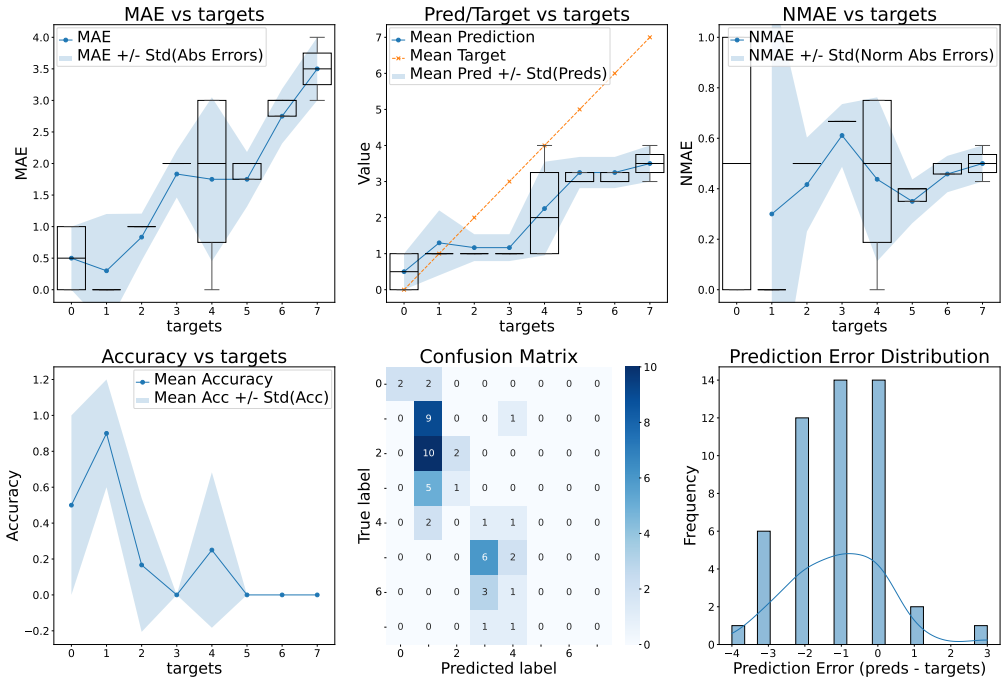


Figure 29: LLaMA-4-Scout results for vertical distance between two pieces on an 8x8 Chess grid. 50 samples overall.

B.8 Identification with camera distance (Chess)

We present the diagnostic results for the *identification of a single piece* task on the Chess dataset, with camera distance increasing. The following question is asked with a debiased preprompt and a declarative instruction:

"This is not a real chess game. The number and position of the pieces can vary arbitrarily. Just focus on answering the following question based on the visual content. What pieces are on the board (among pawn, rook, knight, bishop, king and queen)? Respond in a declarative format."

Examples of images can be found below in Figure 30:

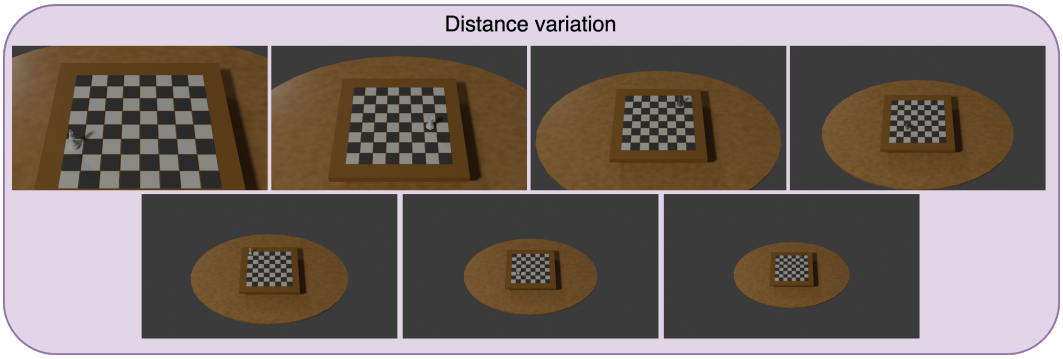
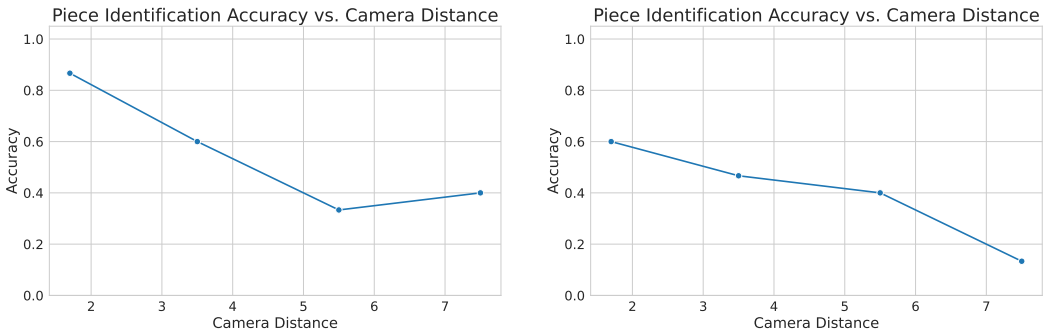


Figure 30: Variations of the 7 distance levels in the chess scene, ranging from closest (top left) to furthest (bottom right), for identifying a single chess piece.

Results (Figures 31b and 31a) As anticipated, the performance of both models declines as the camera distance, i.e. the distance to the chess piece, increases. Nevertheless, GPT-4.1 appears to outperform LLaMA-4-Scout slightly in this task.



(a) GPT-4.1 (b) LLaMA 4

Figure 31: Accuracy for piece type identification at varying camera distances on the Chess dataset, for (a) GPT-4.1 and (b) LLaMA 4. Results are reported over 10 samples per distance level.

B.9 Counting with Overlap (Poker)

We present the diagnostic results for the *counting with overlap* task on the Poker dataset. The following question is asked with a debiased preprompt and a declarative instruction:

"This is not a real poker game. The cards and their position can vary arbitrarily. Just focus on answering the following question based on the visual content. How many cards are on the table? Be aware that some cards might be overlapping. Respond in a declarative format: 'The number of cards on the table is:'"

Examples of images can be found below in Figures 32 and 33:

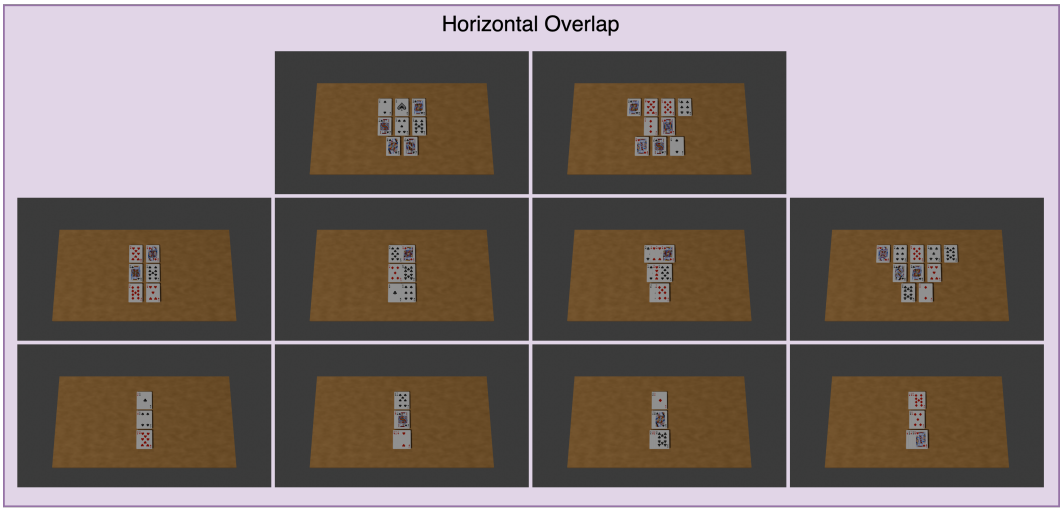


Figure 32: Variations of the horizontal overlap for Poker cards for the *counting with overlap* task

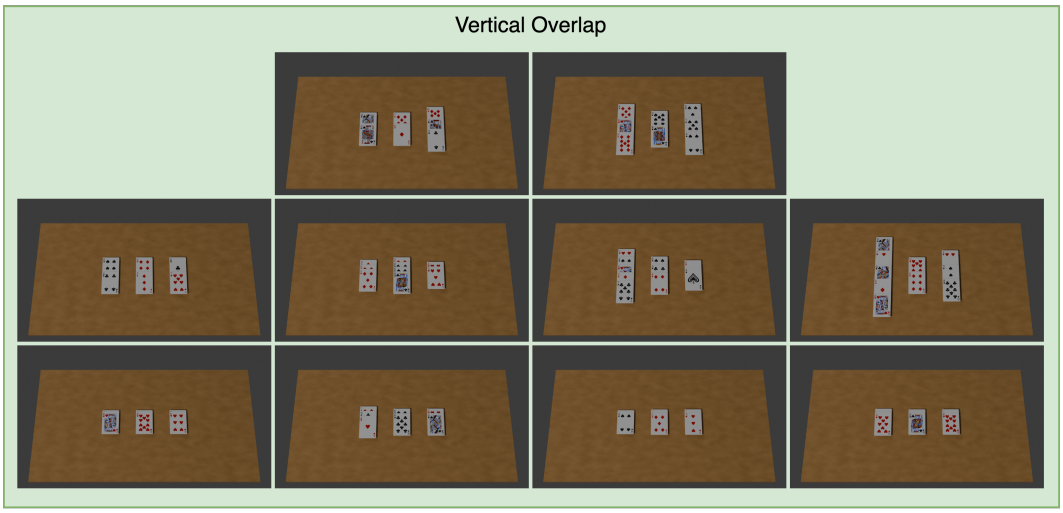


Figure 33: Variations of the vertical overlap for Poker cards for the *counting with overlap* task

Results

- GPT-4.1 (Figures 34,36) As expected, accuracy decreases with increasing horizontal overlap, which reduces the distinguishability of the cards. The model tends to underestimate the number of cards at every level of horizontal overlap. Figure 36 confirms that the difficulty is primarily driven by the overlap, as the highest NMAE values for the highest levels of horizontal overlap occur with a low number of objects

- LLaMA-4-Scout (Figures 35, 36) Accuracy declines as horizontal overlap increases, with the MAE steadily rising alongside the overlap. The model appears to struggle significantly with this aspect, as the difference between the mean prediction and the target grows with increasing overlap, reaching an average discrepancy of up to 5 objects at the highest level. As the overlap between cards increases, the model increasingly underestimates the number of cards, with the distribution of prediction errors clearly shifted to the left and centered around -2. The model also shows a bias toward predicting values of 1-3 cards for each true label, as shown in the confusion matrix, which suggests its difficulty in distinguishing cards that encroach upon each other. Figure 36 further confirms that the primary source of difficulty is the level of overlap, with NMAE increasing as overlap intensifies.

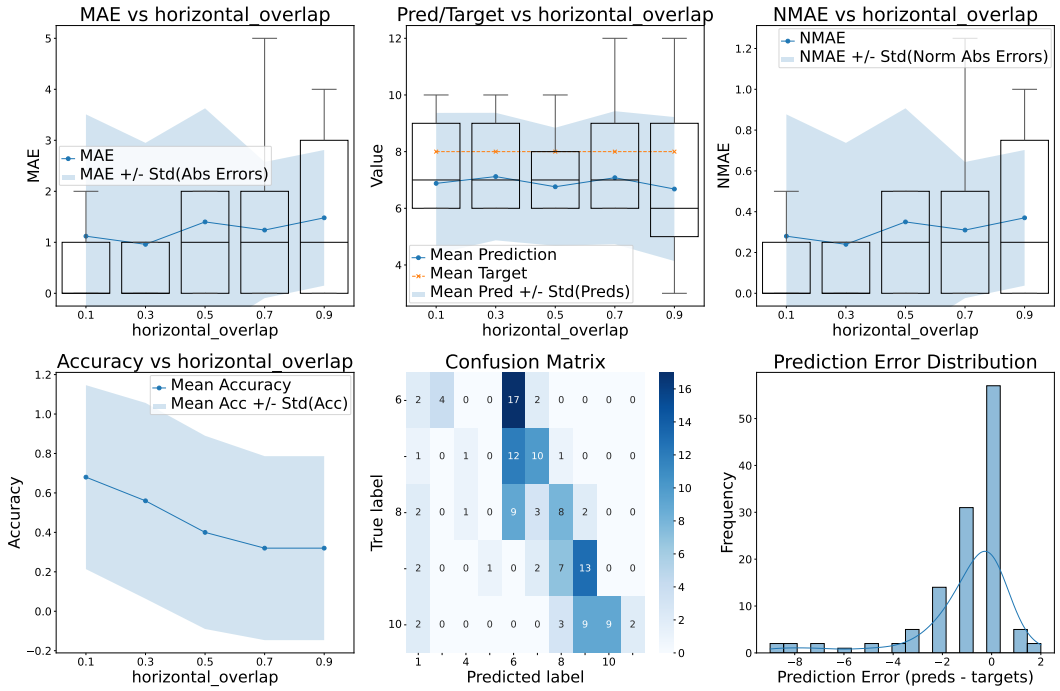


Figure 34: GPT-4.1 results for the counting question under overlapping cards on the Poker dataset. Mean and standard deviation over 5 samples per level.

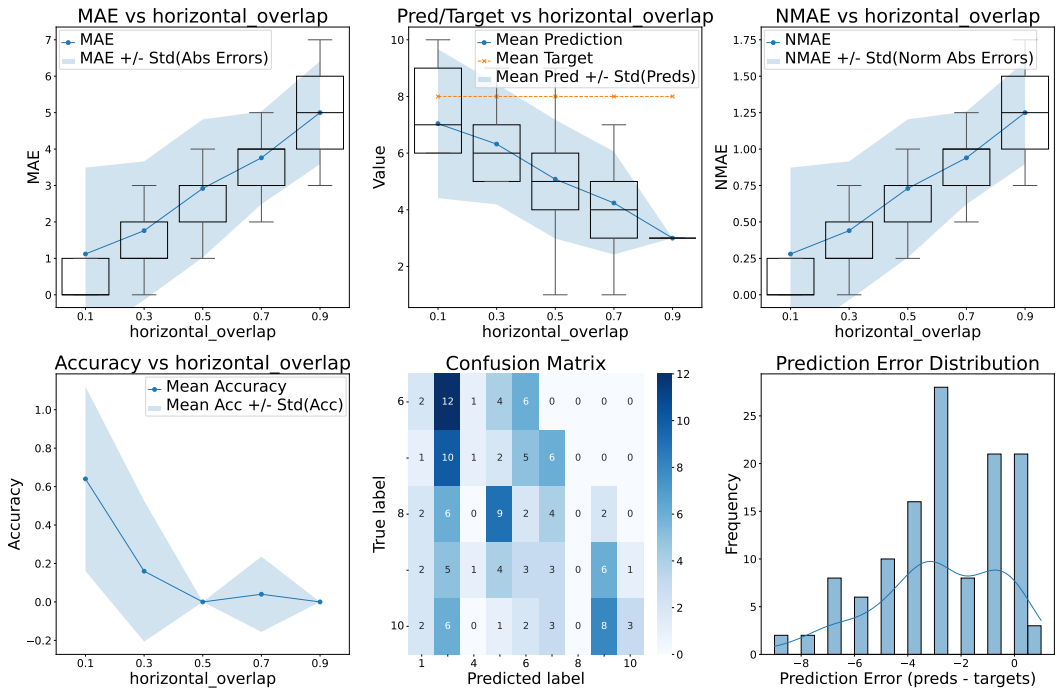
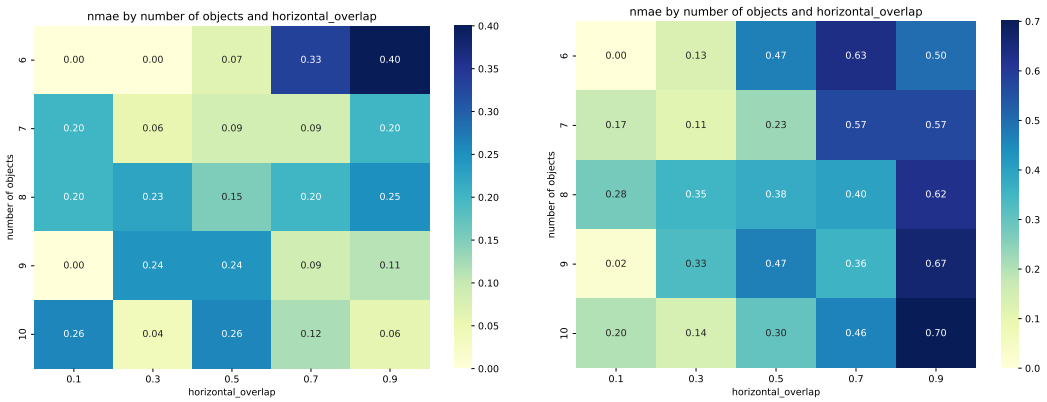


Figure 35: LLaMA results for the counting question under overlapping cards on the Poker dataset. Mean and standard deviation over 5 samples per level.



(a) GPT-4.1 (b) LLaMA-4-Scout

Figure 36: Error analysis for the Poker dataset (overlap counting task): absolute error as a function of overlap level and number of cards, for (a) GPT-4.1 and (b) LLaMA.

C Benchmark - additional experiments

In this section, we present complementary results that strengthen the core findings of the main paper. These additional experiments explore:

- Model robustness under controlled perturbations introduced in synthetic scenes.
- The sensitivity of vision-language models (VLMs) to prompting strategies across tasks.
- Extended comparisons across multiple families of VLMs, including both proprietary (e.g., GPT-4.1) and open-source (e.g., LLaMA-4) models.

All experiments are conducted using a unified evaluation methodology, which ensures consistent sampling, annotation parsing, and metric computation across counting, identification, and localization tasks. This diagnostic protocol enables a fine-grained analysis of model behavior under realistic yet controlled visual conditions.

All experiments are executed using a unified evaluation pipeline, which integrates:

- Standardized data ingestion from our diagnostic dataset, including task-specific image folders and structured annotations,
- Modular VLM querying interfaces supporting OpenAI, Groq, and OLLaMA endpoints,
- A decoding and scoring pipeline tailored to each task type (e.g., MAE for counting, token-level F1 for identification, accuracy for localization),
- Reproducible configuration management, enabling consistent experimental tracking across noise levels, prompt variations, and model families.

This diagnostic setup enables a fine-grained understanding of model generalization under distribution shifts that are difficult to control in real-world data. The additional results presented below confirm the necessity of such synthetic stress tests in assessing the real-world readiness of multimodal models.

We include below additional heatmaps generated for each model across specific diagnostic task variants. These highlight the spatial structure of predictions and common failure modes under controlled synthetic conditions. Each heatmap corresponds to a specific combination of prompt strategy (Helpful, Neutral, Chain-of-Thought [61]) and prompt reformulation (Declarative or Missing Word). Figure 42 illustrates model outputs for a representative example of the counting task.

C.1 Count Blur

This diagnostic setup introduces the **count blur** variation, where we simulate depth-of-field effects by applying spatially localized camera blur to the input images. This mimics real-world conditions such

as defocus, motion blur, or background noise, which often reduce the clarity of visual cues necessary for accurate object enumeration. The primary objective of this variant is to assess the robustness of Vision-Language Models (VLMs) in scenarios where object boundaries become ambiguous or occluded due to degraded image quality. By varying the intensity and location of the blur (e.g., central vs. peripheral regions), we can analyze whether models rely on holistic scene understanding or overfit to sharp, high-frequency regions. This setup serves as a stress test for the model's spatial reasoning and ability to generalize counting abilities beyond clean synthetic. Figures 37 and 38 illustrate respectively the results for Chess and Poker datasets. The Table 5 highlights the results of the models on different blur conditions.

Table 4: VLM scores over the counting task on Chess dataset (↑: higher is better, ↓: lower is better). Declarative reformulation, helpful preprompt. Piece count ranges from 1 to 21.

Model	Acc↑	F1↑	Prec↑	Rec↑	MAE↓	MSE↓	NMAE↓	Rel↑	Cons↑
gpt-4.1	0.743	0.853	0.793	0.922	0.319	0.476	0.160	0.743	0.262
gpt-4.1-mini	**0.806**	**0.893**	**0.828**	**0.969**	**0.209**	**0.251**	**0.084**	**0.806**	0.277
LLaMA-4-scout	0.639	0.780	0.678	0.917	0.466	0.707	0.202	0.639	0.267
LLaMA-4-maverick	0.445	0.616	0.494	0.817	0.670	0.921	0.318	0.445	0.330
gemma3:4b	0.476	0.645	0.503	0.901	0.838	1.906	0.317	0.476	0.419
gemma3:12b	0.495	0.645	0.543	0.916	0.862	1.893	0.320	0.487	0.459
LLaMA3.2-vision	0.400	0.571	0.455	0.857	0.932	2.356	0.590	0.403	0.343
mistral-small3.1	0.520	0.684	0.520	0.923	0.880	1.802	0.224	0.536	**0.557**

Each heatmap reflects the outcome of a specific preprompt and reformulation strategy combination. We compare:

- **Prompt strategies:** Helpful, Neutral, and Chain-of-Thought (CoT).
- **Reformulations:** Declarative vs. Missing Word.

Notable observations:

- Helpful + Declarative combinations consistently yield the most accurate and spatially uniform predictions, especially for GPT-4.1.
- Missing Word formulations are more brittle under blur, leading to degraded performance even for strong models.
- LLaMA and Gemma models show inconsistent spatial activation, especially near the image center, indicating struggles with object separation when blur is present.

In the figure 37 GPT models demonstrate strong robustness under blur, achieving the best accuracy under Declarative reformulation and Helpful preprompt. Their performance remains stable even with central blur, highlighting effective abstraction over degraded input features. LLaMA-4-Scout achieves surprisingly strong results with Declarative reformulation and Helpful preprompt but collapses under Missing Word reformulation, indicating high sensitivity to linguistic scaffolding. LLaMA-4-Maverick and Gemma3 exhibit irregular spatial activation patterns and struggle with object separation under blur. Their heatmaps often show fragmented or inconsistent predictions, especially near the image center, where blur is typically applied.

In contrast to the chess setup, the Poker Count Blur results (Figure 38) reveal more pronounced variability across models and prompting strategies. While GPT models again lead overall performance, their spatial activation maps suggest slightly reduced robustness compared to the chess scenes, likely due to the more irregular and less structured layout of poker elements. GPT-4.1, in particular, maintains high spatial accuracy across all prompt variations, with strong resilience even under significant central blur.

LLaMA-4-Maverick performs surprisingly well in this setup, showing more consistent activation than in the chess domain, and demonstrating better object separation under blur. This suggests the model benefits from the sparser spatial distribution of poker tokens. LLaMA-4 Scout also achieves strong performance when guided by Declarative + Helpful prompts, but, similar to the chess task, degrades sharply under Missing Word configurations.

Gemma3:12b continues to underperform across all prompting strategies, with heatmaps revealing scattered and incoherent predictions, particularly in blurred central zones. Compared to chess, the poker domain further exposes the model's limited robustness and poor generalization under visual uncertainty.

As with the chess task, prompting strategy plays a pivotal role. Declarative prompts combined with Helpful scaffolding consistently yield the most spatially stable predictions. In contrast, Chain-of-Thought reasoning again appears to destabilize performance—likely due to irrelevant or hallucinated intermediate reasoning steps that fail to compensate for degraded visual cues.

Prompting strategies play a critical role. Declarative prompts consistently outperform Missing Word formulations across all models. Chain-of-Thought tends to destabilize performance in this setup, likely due to added reasoning steps that are less effective when visual cues are ambiguous. In Table 4 we illustrate the results for the Counting task on 800 samples of the Chess Dataset.

The results presented in Figure 39 offer a detailed breakdown of model performance under the Count Blur setting across two tasks (chess and poker), and two linguistic dimensions: preprompt type and reformulation style.

Across all conditions, GPT-based models (GPT-4.1 and GPT-4.1-mini) demonstrate superior F1 scores, particularly under the Helpful + Declarative combination. This pattern is consistent in both structured (chess) and unstructured (poker) scenes, suggesting strong robustness to visual degradation when linguistic scaffolding is explicit. GPT-4.1-mini slightly trails GPT-4.1 but generally follows similar trends, indicating that model size may affect resilience only marginally when prompts are clear.

In contrast, open-source models such as LLaMA and Gemma display greater variance in F1 performance, underscoring their higher sensitivity to prompt phrasing. For example, LLaMA-4-Scout performs competitively in declarative setups but exhibits sharp drops with Missing Word reformulations, particularly in poker scenes where spatial regularity is lacking. This drop reveals a prompt-induced brittleness in models without strong pretrained instruction alignment.

Interestingly, LLaMA-4-Maverick shows more consistent performance in the poker task than in chess. One hypothesis is that in less structured layouts (like poker), the model may rely more on global object counting rather than exact spatial localization, reducing its reliance on precise blur-resilient visual grounding.

The bar plots also reinforce that Chain-of-Thought reasoning is not universally helpful: while it benefits certain configurations, it often destabilizes performance in blur-sensitive contexts. This is likely due to additional reasoning steps introducing error propagation when visual inputs are already degraded.

These findings reinforce a key insight from our main experiments: prompting design plays a pivotal role in model robustness, especially under partial information scenarios. Declarative prompts and helpful framing not only improve raw accuracy but also contribute to spatial consistency and reduced confusion under blur conditions.

The full interaction plots in Figures 40 and 41 expose how prompt type and reformulation style interact to amplify or mitigate performance gaps under Blur conditions. In both domains, Helpful preprompt with Declarative reformulation combination remains the clear optimum, driving GPT to near-ceiling F1 scores. However, the penalty for moving away from this configuration is notably task-dependent:

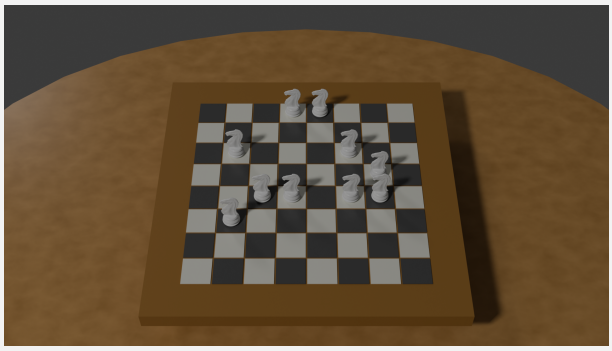
- **Poker scenes** (Figure 40) show the widest spread across bars, with open-source models collapsing under Neutral preprompt with Missing-Word reformulation, or even CoT preprompt with Missing Word reformulation, highlighting their reliance on explicit guidance when object layout is irregular.
- **Chess scenes** (Figure 41) exhibit a narrower spread, yet still reveal a consistent drop for LLaMA and Gemma variants whenever either the preprompt is less informative or the reformulation omits key lexical cues.

Taken together, these interaction plots underscore the compounding effect of linguistic scaffolding on model robustness. Our findings suggest that both preprompt framing and reformulation structure contribute synergistically to performance under visual degradation. The absence of either component,

Table 5: Performance Metrics for Count Blur Task. Performance on the counting task with different levels of blur. Low, Medium and High blur conditions progressively reduce visibility.

Blur Condition	Model	Accuracy (%)	MAE
Low Blur: 1	gpt-4.1	74.3	0.32
	gpt-4.1-mini	**80.6**	**0.21**
	LLaMA-4-scout-17b-16e-instruct	63.9	0.47
	LLaMA-4-maverick-17b-128e-instruct	44.5	0.67
	gemma3:12b	45.0	1.41
Medium Blur: 3	gpt-4.1	74.0	0.33
	gpt-4.1-mini	**76.0**	**0.25**
	LLaMA-4-scout-17b-16e-instruct	59.0	0.54
	LLaMA-4-maverick-17b-128e-instruct	40.0	0.69
	gemma3:12b	39.0	1.59
High Blur: 5	gpt-4.1	**55.0**	**0.69**
	gpt-4.1-mini	51.0	1.40
	LLaMA-4-scout-17b-16e-instruct	25.0	8.23
	LLaMA-4-maverick-17b-128e-instruct	40.0	5.74
	gemma3:12b	26.0	4.58

particularly in open-source models with limited instruction tuning, leads to a marked decline in task reliability, highlighting the necessity of jointly optimizing prompt clarity and linguistic structure in VLM evaluation protocols.



Input Image

Task: Number Pieces Counting in Synthetic Chessboard Image
Question: Analyze the image carefully, be aware that this image may differ from a classical chessboard scene. How many pieces are there in the image? Respond in a declarative format: *"The number of pieces in the image is:"*

Model	Response	
GPT-4.1	The number of pieces in the image is: **10**	✓
GPT-4.1-mini	The number of pieces in the image is: **10**	✓
LLaMA-4-scout	The number of pieces in the image is: **8**	✗
LLaMA-4-maverick	The number of pieces in the image is: **9**	✗
Gemma-4b	The number of pieces in the image is: **8**	✗
Gemma-12b	The number of pieces in the image is: **8**	✗
LLaMA3.2-vision	The number of pieces in the image is: **12**	✗
Mistral-small3.1	The number of pieces in the image is: **10**	✓

Ground Truth: 10

Figure 42: VLM responses to a number pieces counting question based on a synthetic chess scene.

C.2 Identification

This task assesses a model's ability to recognize and classify individual chess pieces in cluttered scenes where objects are tightly packed. Unlike counting or localization, the identification task demands precise type-level resolution (e.g., distinguishing a white pawn from a black rook) based on both visual appearance and spatial cues. The setup is particularly challenging as it stresses the model's fine-grained visual understanding, symbol-to-label mapping, and resistance to object occlusion or visual overlap.

Models are evaluated across variations in prompt formulation (Declarative vs. Missing Word) and instruction strategy (helpful, Chain-of-Thought, Neutral). This allows us to test how linguistic scaffolding influences the model's ability to resolve ambiguous visual identities.

Insights:

- Declarative reformulations enhance type discrimination, especially when paired with explicit prompts (e.g., "Identify all...").
- Chain-of-Thought prompts do not consistently improve results in this task, possibly due to hallucinated intermediate steps.
- Open models (e.g., Gemma, Mistral) exhibit frequent misidentifications in cluttered areas regardless of prompting, indicating limited visual resolution or weak grounding.

Overall, GPT-4.1-mini demonstrates the most consistent performance across all prompting styles, with the highest accuracy observed with Declarative reformulation and Helpful preprompt. Its heatmap shows well-defined predictions with minimal confusion between piece types. Mistral-small3.1 performs moderately well suggesting relatively stable visual grounding, though it suffers slightly more in scenes with high visual congestion. LLaMA variants show mixed behavior. Accuracy range depends on prompt configuration. Notably, LLaMA-4 Scout variant briefly peaks with Declarative reformulation and Helpful preprompt, indicating some sensitivity to instruction quality. Gemma models underperform consistently, likely due to limited visual fidelity and weaker object-label grounding. They frequently confuse similar-looking pieces and fail to disambiguate based on position or color cues. Across all models, Declarative reformulations with Helpful preprompting result in the most accurate and spatially stable identification. Chain-of-Thought prompts often lead to hallucinated intermediate reasoning, especially for open-source models, reducing final accuracy.

C.3 Identification Distance

This variation probes the ability of models to identify object types when the objects are spaced apart across the scene. Unlike the standard Identification setup where objects are densely packed, this version introduces increasing spatial distance between entities. This setting reduces visual occlusion but increases the need for robust spatial reference and positional grounding.

It simulates real-world perception scenarios such as autonomous navigation or robotic sorting tasks, where entities must be recognized from a distance or across sparsely populated environments.

We again compare different prompting styles (Helpful, Chain-of-Thought, Neutral) and reformulation templates (Declarative vs. Missing Word), focusing on how they modulate model attention and symbol-to-object mapping under reduced visual clutter. Figure 44 illustrates the different results.

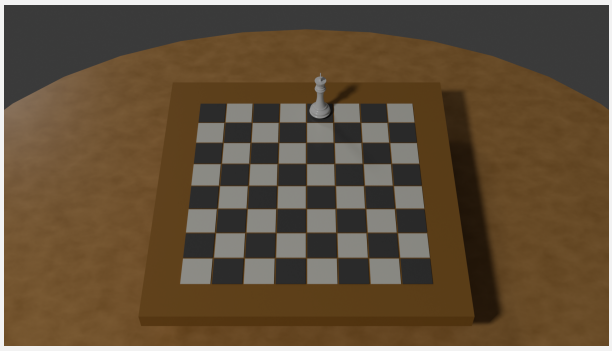
The Poker Identification Distance results further highlight the role of linguistic scaffolding in modulating VLM behavior under low-density, spatially separated settings. As shown in Figure 45, GPT-based models maintain strong performance across all prompting styles, with the best F1 scores observed under the Declarative + Helpful configuration. This suggests that these models are capable of leveraging both spatial context and linguistic cues to maintain object identity across distributed scenes, a property especially useful in real-world applications like robotic vision or inventory inspection.

Interestingly, LLaMA-4 Maverick outperforms Scout in most configurations for this task, reflecting improved stability under distance-based separation. However, both models remain highly sensitive to prompt phrasing: performance drops significantly when the prompt becomes less explicit, as in the Missing Word condition. Gemma's results confirm this fragility; it shows erratic responses and reduced F1 scores across both prompting and reformulation variations.

Notably, while the overall visual complexity is lower in this task compared to densely packed scenes, prompting style still significantly affects performance. Chain-of-Thought reasoning, in particular, continues to be a double-edged sword: in some cases, it helps weaker models by injecting reasoning structure, but often it introduces noise that leads to reduced confidence in symbol-to-object resolution. These findings confirm that even when objects are well-separated, linguistic design remains a critical lever for guiding model precision in identity-based tasks. Figure 46 presents a representative example of model predictions for the identification task.

Findings:

- GPT-family models leverage spatial separation well, especially under Helpful prompts with Declarative phrasing.
- LLaMA-based models underperform in Declarative + CoT combinations, possibly due to misalignment between reasoning chains and visual grounding.
- Missing Word prompts reduce recall at peripheral regions, indicating sensitivity to template ambiguity.



Input Image

Task: Color Identification in Synthetic Chessboard Image
Question: Analyze the image carefully. What color is the piece on the board? Respond in a declarative format: *"The color of the piece on the board is:"*

Model	Response	
GPT-4.1	The color of the piece on the board is: **white**	✓
GPT-4.1-mini	The color of the piece on the board is: **white**	✓
LLaMA-4-scout	The color of the piece on the board is: **white**	✓
LLaMA-4-maverick	The color of the piece on the board is: **white**	✓
Gemma-4b	The color of the piece on the board is: **white**	✓
Gemma-12b	The color of the piece on the board is: **white**	✓
LLaMA3.2-vision	The color of the piece on the board is: **white**	✓
Mistral-small3.1	The color of the piece on the board is: **white**	✓

Ground Truth: White
Result: All models answered correctly.

Figure 46: VLM responses to a color identification question based on a synthetic chess scene.

C.4 Localization

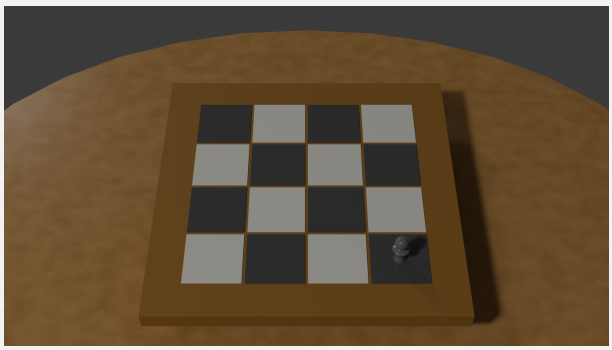
This diagnostic task evaluates the ability of the models to localize individual objects within compact and discrete spatial layouts. In the **Chess** setting, models are required to identify the position of specific pieces on a board where even minor spatial drift leads to errors. This setup emulates constrained applications such as tabletop manipulation or industrial assembly tasks, where visual clutter is minimal but spatial precision is critical.

In the **Poker** setup, we use a 3x3 grid containing a single visible card per scene. This configuration evaluates localization under lower density but higher ambiguity due to card orientation, subtle visual differences, and less structured spatial priors.

Unlike standard object detection tasks, this diagnostic focuses on fine-grained spatial grounding: the model must correctly associate linguistic expressions (e.g., "the red king in the bottom left") with discrete grid locations. We vary both *preprompt strategies* (Helpful, Chain-of-Thought, Neutral) and *reformulation types* (Declarative vs. Missing Word) to assess how instruction framing affects spatial accuracy. Figure 47 presents an example showcasing the predictions of different models on a localization task sample. **Key findings:**

- **GPT-based models** (particularly GPT-4.1 and GPT-4.1-mini) demonstrate high spatial precision and uniform activation across grid cells, especially when prompted with Declarative + Helpful combinations.

- **Chain-of-Thought prompting** introduces variance in localization, often disrupting spatial alignment by triggering non-grounded reasoning pathways.
- **Open-source models** (e.g., LLaMA, Gemma) are more sensitive to instruction type. LLaMA variants exhibit degraded and inconsistent localization under blur or ambiguous phrasing; Gemma shows highly noisy and scattered predictions across all prompting styles.



Input Image

Task: Piece Localization in Synthetic Chessboard Image
Column Question: Analyze the image carefully, be aware that this image may differ from a classical chessboard scene. Numbering the columns from left to right, starting with 0, on which column is the piece on the board? Respond in a declarative format: *"The column on which the piece is on the board is:"*
Row Question: Analyze the image carefully, be aware that this image may differ from a classical chessboard scene. Numbering the rows from top to bottom, starting with 0, on which row is the piece on the board? Respond in a declarative format: *"The row on which the piece is on the board is:"*

Model	Response	
GPT-4.1	The column on which the piece is on the board is: **3**	✓
GPT-4.1	The row on which the piece is on the board is: **3**	✓
GPT-4.1-mini	The column on which the piece is on the board is: **3**	✓
GPT-4.1-mini	The row on which the piece is on the board is: **3**	✓
LLaMA-4-scout	The column on which the piece is on the board is: **2**	✗
LLaMA-4-scout	The row on which the piece is on the board is: **3**	✓
LLaMA-4-maverick	The column on which the piece is on the board is: **3**	✓
LLaMA-4-maverick	The row on which the piece is on the board is: **3**	✓
Gemma-4b	The column on which the piece is on the board is: **2**	✗
Gemma-4b	The row on which the piece is on the board is: **1**	✗
Gemma-12b	The column on which the piece is on the board is: **3**	✓
Gemma-12b	The row on which the piece is on the board is: **3**	✓
LLaMA3.2-vision	The column on which the piece is on the board is: **2**	✗
LLaMA3.2-vision	The row on which the piece is on the board is: **2**	✗
Mistral-small3.1	The column on which the piece is on the board is: **3**	✓
Mistral-small3.1	The row on which the piece is on the board is: **3**	✓

Ground Truth Column: 3
Ground Truth Row: 3

Figure 47: VLM responses to a color localization question based on a synthetic chess scene.

The Figure 48 illustrates the different results. Despite its smaller scale relative to GPT-4 models, **Mistral-small3.1** delivers surprisingly competitive performance on the 4×4 localization task. Its

heatmaps reveal coherent spatial activation with low positional drift across most prompting conditions. While it does not achieve the spatial uniformity of GPT-based models, Mistral demonstrates reliable grounding under Declarative + Helpful prompts and maintains acceptable coverage even with Neutral inputs. However, mild instability emerges under Chain-of-Thought prompting, where reasoning steps appear to interfere with direct spatial mapping—occasionally resulting in diagonal mislocalizations. These results suggest that Mistral possesses a moderately strong spatial prior and can generalize across linguistic formats, though it remains more sensitive to prompt structure than instruction-tuned proprietary models.

By contrast, **LLaMA-4 Scout and Maverick** exhibit degraded and inconsistent spatial activation. Their performance is highly contingent on explicit prompt framing, with significant accuracy drops under less structured instructions. The **LLaMA-3.2 Vision** variant performs worst overall: its heatmaps frequently contain empty or incoherent activations, indicating a fundamental deficiency in visual-spatial alignment despite its multimodal architecture.

Table 6: VLM scores over the relative localization task on the Chess dataset, with declarative instruction and helpful preprompt (↑: higher is better, ↓: lower is better. The grid is a 4×4 Board.

Model	**Acc↑**	**F1↑**	**Prec↑**	**Rec↑**	**MAE↓**	**MSE↓**	**NMAE↓**
gpt-4.1	**0.790**	**0.883**	**0.798**	**0.988**	**0.210**	**0.210**	**0.097**
gpt-4.1-mini	0.570	0.726	0.704	0.750	0.480	0.600	0.382
LLaMA-4-scout	0.290	0.450	0.382	0.547	2.580	13.96	1.693
LLaMA-4-maverick	0.510	0.675	0.671	0.680	0.550	0.690	0.440
gemma3	0.420	0.592	0.447	0.875	0.760	1.120	0.445
gemma3:12b	0.170	0.291	0.283	0.298	1.320	2.520	0.947
LLaMA3.2-Vision	0.230	0.374	0.411	0.343	0.940	1.280	0.665
mistral-3.1	0.540	0.701	0.675	0.730	0.520	0.660	0.382

In Figure 49, we focus on two representative prompting configurations: **Declarative + Helpful** and **Missing Word + Helpful**. This choice reflects prior findings across our evaluation suite, where Declarative + Helpful consistently yielded the highest spatial accuracy, while Missing Word + Helpful exposed prompt-induced variability.

Results on the poker localization task confirm this trend. GPT-based models (GPT-4.1 and GPT-4.1-mini) achieve top-tier F1 scores under both conditions, demonstrating robustness to reformulation style. Mistral-small3.1 performs reasonably well under explicit prompting but exhibits degradation with Missing Word formulations, suggesting limited resilience to implicit language structures.

In contrast, LLaMA variants particularly Scout and Vision experience sharp performance drops when explicit lexical cues are removed, reinforcing their reliance on surface prompt clarity. Gemma models perform weakest overall, struggling even under the most favorable conditions. These results emphasize that even in low-density visual environments, prompt design remains a dominant factor influencing spatial reasoning, especially for models with weaker instruction tuning.

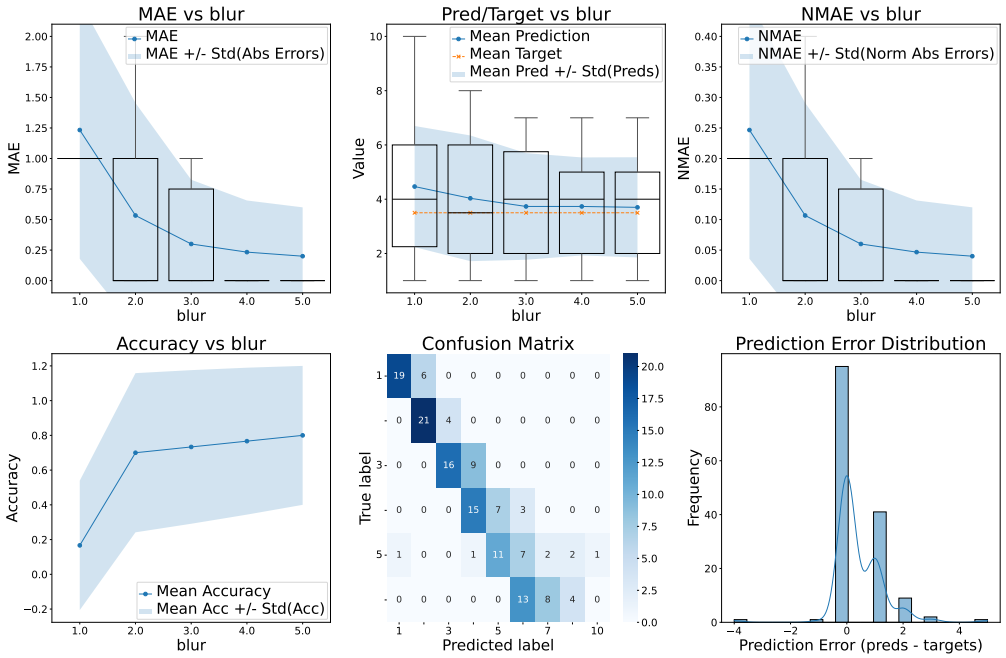


Figure 14: Results of GPT-4.1 for the counting question under blur decrease (higher values meaning less blur) on the Chess dataset. Mean and standard deviation over 5 levels (from 1 to 5), with 5 samples per level.

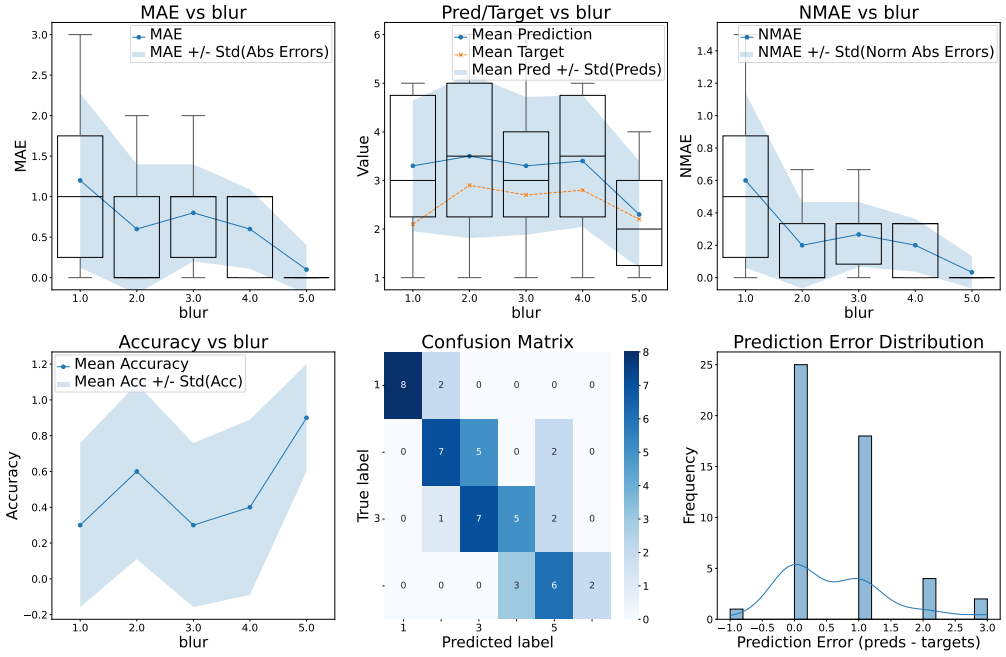


Figure 15: Results of LLaMA-4-Scout for the counting question under blur decrease (higher values meaning less blur) on the Chess dataset. Mean and standard deviation over 5 levels (from 1 to 5), with 5 samples per level.

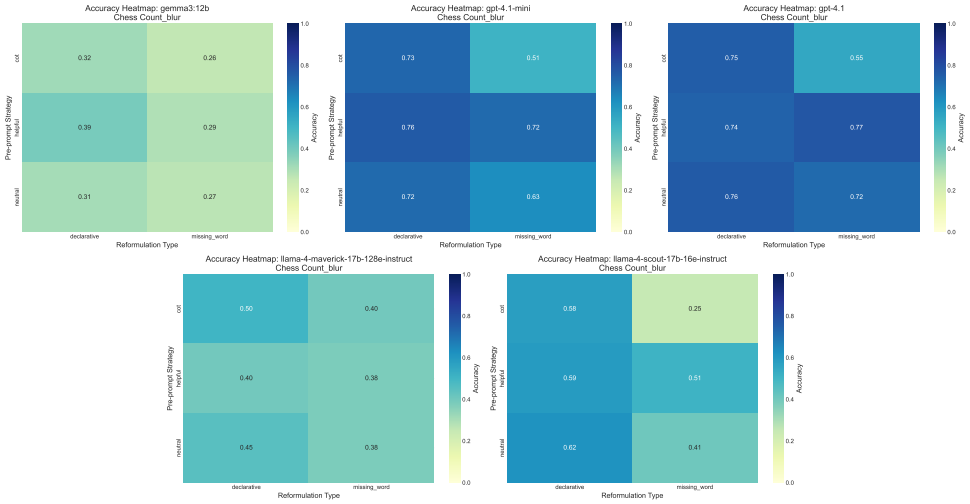


Figure 37: Spatial accuracy heatmaps for the Chess Count Blur task across multiple models and prompting strategies, averaged over 191 synthetic chess scenes with localized camera blur. Declarative prompts paired with Helpful strategies yield the most spatially stable and robust performance under visual degradation.

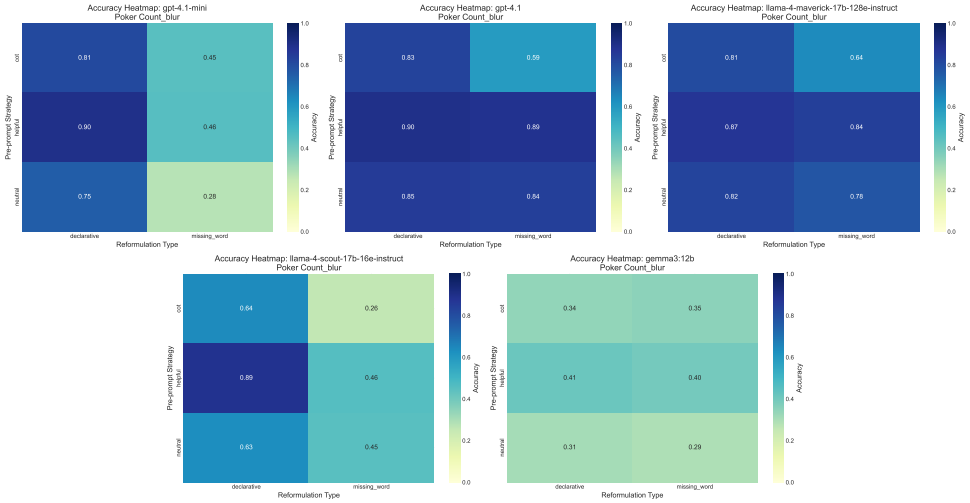
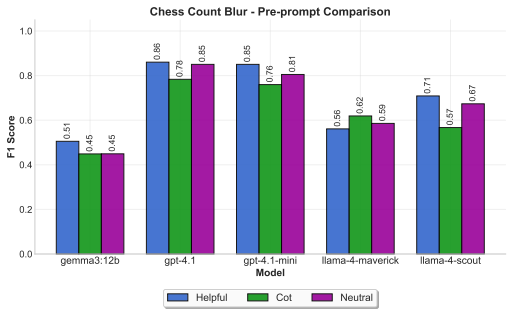
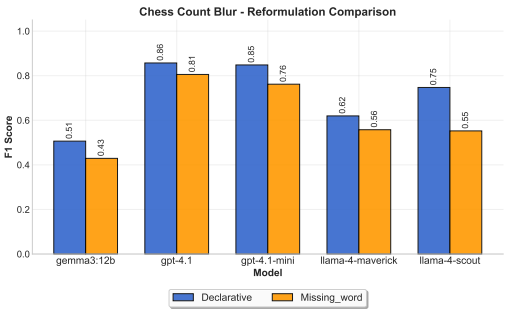


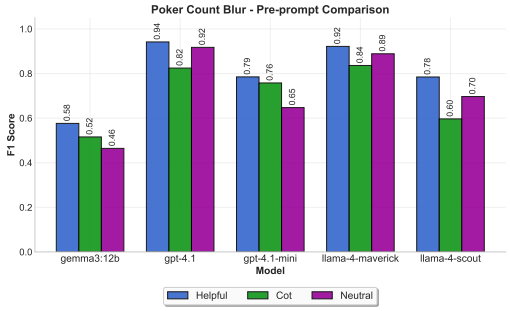
Figure 38: Spatial accuracy heatmaps for the Poker Count Blur task across multiple models and prompting strategies, averaged over 50 synthetic poker scenes with localized camera blur. Declarative prompts paired with Helpful strategies yield the most spatially stable and robust performance under visual degradation.



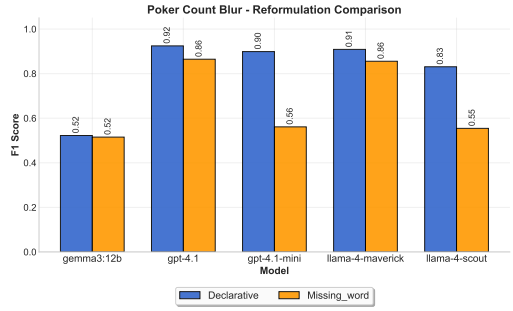
(a) Chess – preprompt Comparison



(b) Chess – Reformulation Comparison



(c) Poker – preprompt Comparison



(d) Poker – Reformulation Comparison

Figure 39: **F1 Score Comparison for Count Blur Tasks in Chess and Poker.** This figure shows how linguistic strategies affect model performance under blurred visual conditions. (a) and (c) compare preprompt types (Helpful, Chain-of-Thought, Neutral), while (b) and (d) compare reformulation styles (Declarative vs. Missing Word). GPT models consistently benefit from explicit linguistic scaffolding, while LLaMA and Gemma variants show significant variance depending on prompt configuration.

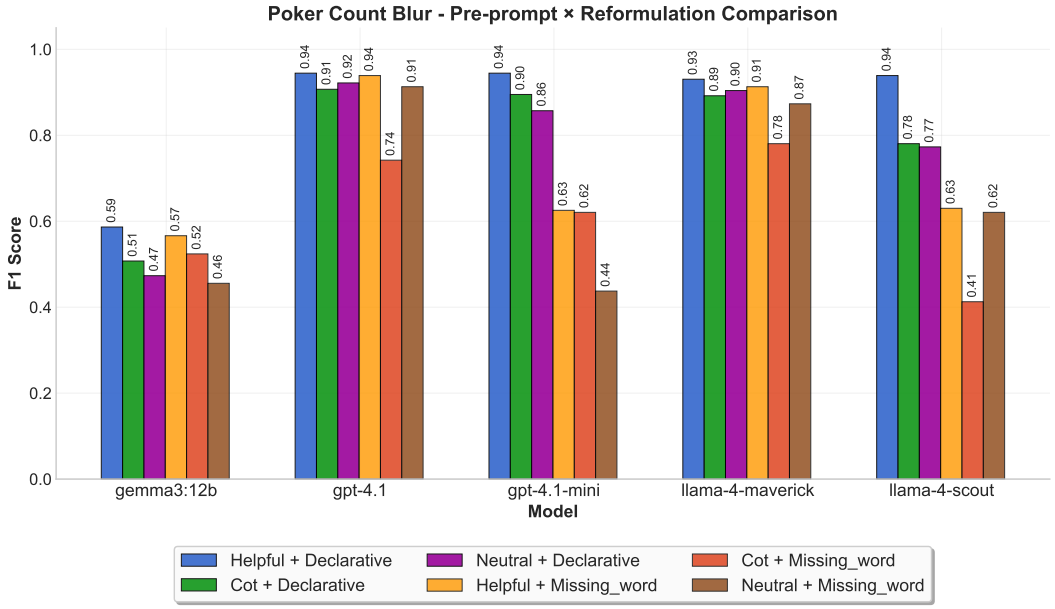


Figure 40: **F1 Score Comparison for Count Blur (Poker) across Prompt × Reformulation Combinations.** The interaction of prompting strategies reveals similar trends as in the chess setting, but with even sharper performance gaps between configurations. GPT-4.1 achieves near-ceiling F1 scores under Helpful + Declarative input, while open-source models such as Gemma and LLaMA degrade significantly when phrasing is ambiguous or under-specified (e.g., Neutral + Missing Word). The broader spread across combinations emphasizes the importance of carefully aligned instructions for models operating under visual degradation.

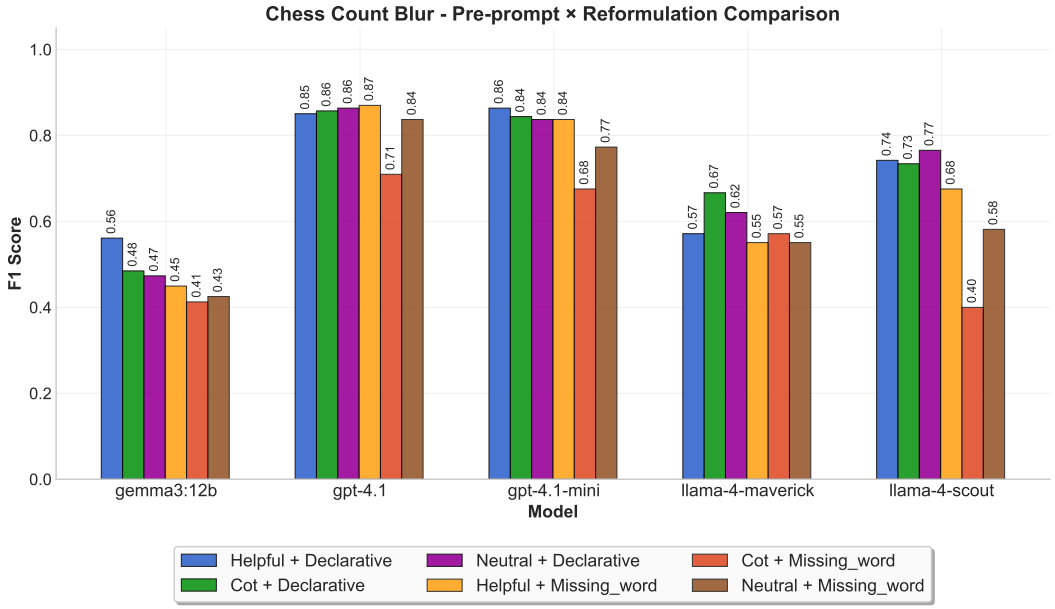


Figure 41: **F1 Score Comparison for Count Blur (Chess) across Prompt × Reformulation Combinations.** Each bar represents model performance under a unique combination of preprompt style (Helpful, CoT, Neutral) and reformulation strategy (Declarative, Missing Word). GPT models (GPT-4.1, GPT-4.1-mini) perform robustly across most configurations, especially with Helpful + Declarative prompting. In contrast, LLaMA and Gemma exhibit strong variability—particularly under Missing Word reformulations—showing reduced reliability when linguistic clarity decreases.

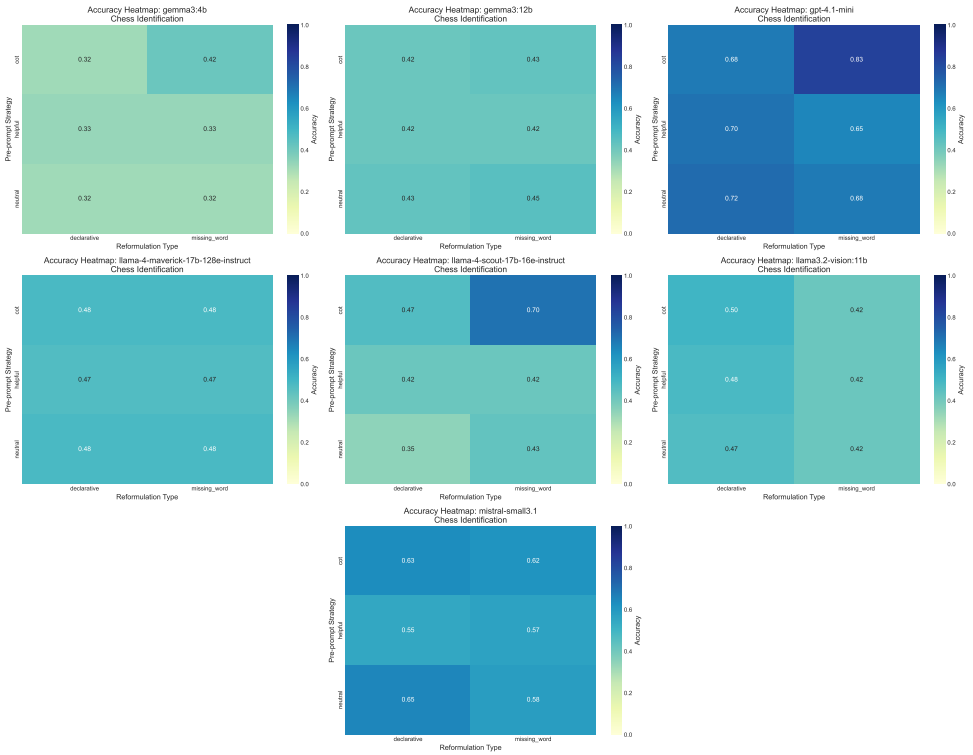


Figure 43: Heatmaps for the Identification task across multiple models and prompting strategies, averaged over 30 densely packed synthetic chess scenes.

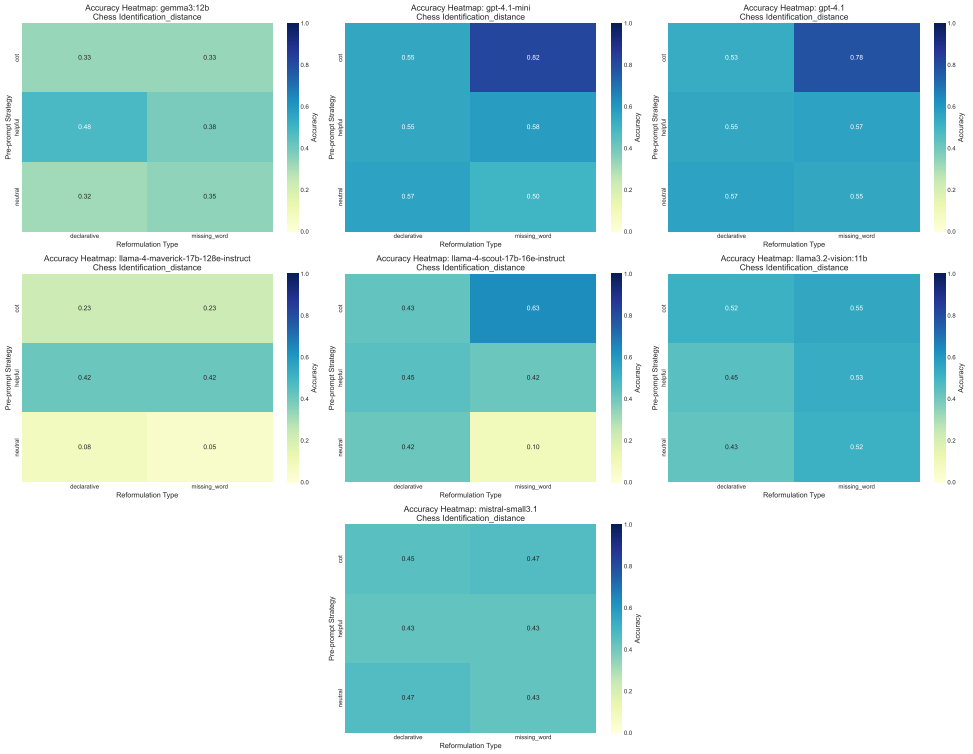
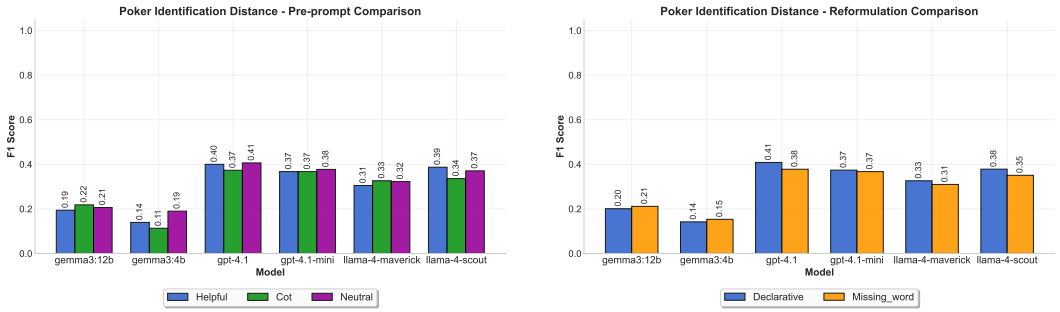


Figure 44: Heatmaps for the Identification Distance task across multiple models and prompting strategies, averaged over 140 spatially-separated synthetic scenes. This variation isolates long-range visual discrimination capacity.



(a) Preprompt Comparison **(b)** Reformulation Comparison

Figure 45: **F1 Score Comparison for the Poker Identification Distance Task.** This figure presents a detailed comparison of model performance when identifying spatially-separated objects in poker scenes under varied linguistic conditions. Subfigure (a) compares preprompt types (Helpful, Chain-of-Thought, Neutral), while subfigure (b) contrasts reformulation styles (Declarative vs. Missing Word). GPT models consistently benefit from explicit linguistic scaffolding, particularly the Declarative + Helpful configuration, achieving higher F1 scores and more stable outputs. In contrast, open-source models (LLaMA, Gemma) exhibit greater sensitivity to prompt clarity and structure, with sharp performance drops under ambiguous phrasing. These results reinforce that even in low-occlusion, high-separation contexts, prompt design remains essential for achieving reliable identification performance.

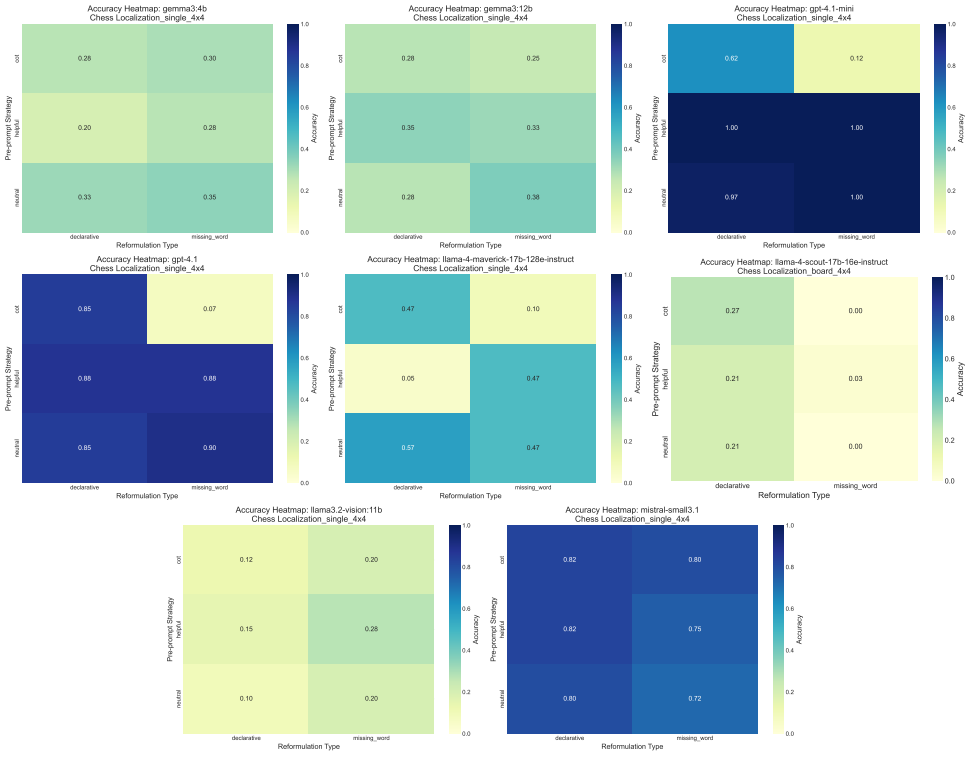


Figure 48: Heatmaps for the Localization Board 4x4 task across multiple models and prompting strategies, averaged over 50 synthetic chess scenes. Bright areas indicate accurate localization; darker areas highlight regions of frequent misprediction. This diagnostic reveals how well models spatially ground objects under minimal layout constraints.

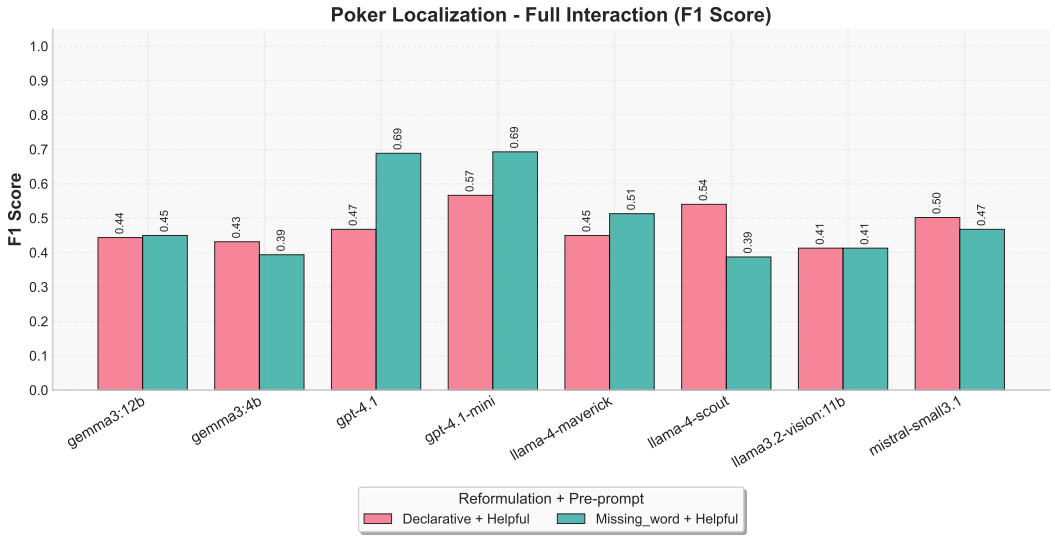


Figure 49: **F1 Score for Poker Localization Task across Prompt × Reformulation Combinations.** This figure visualizes the effect of linguistic scaffolding on model performance in the poker localization setting (3×3 grid). Bars represent F1 scores for each model under two prompting strategies: Declarative + Helpful and Missing Word + Helpful. GPT models (notably GPT-4.1 and GPT-4.1-mini) achieve the highest accuracy, while open-source models exhibit varying sensitivity to reformulation style.

D Detailed Framework

This section provides a comprehensive description of the dataset generation framework developed for structured visual reasoning tasks, with a particular focus on chess and poker scenes. Our framework is designed for maximal modularity, reproducibility, and flexibility: it supports precise configuration of scene elements, systematic variation of experimental parameters, and rigorous annotation for downstream evaluation. We first detail the process for image dataset generation, from scene construction in Blender through to the integration of controlled noise and domain-specific logic for chess and poker. We then describe the formulation and handling of questions and answers, and finally the mechanisms for result retrieval and model evaluation.

D.1 Image dataset generation

The generation of image datasets in our framework is anchored in a modular, scriptable pipeline built on Blender. We begin by designing a scene configuration that specifies all key visual and physical parameters—ranging from object placement, camera setup, lighting, and materials, to controlled sources of variability and noise. Scene elements are modeled as strongly-typed configuration objects, ensuring both clarity and extensibility. The framework accommodates both procedural content generation (directly via Blender's Python API) and the integration of pre-designed assets, and it exposes multiple layers for customization—from low-level mesh manipulation to high-level experiment orchestration. The following subsections describe the main components of this pipeline, starting with the foundational capabilities and scripting interfaces provided by Blender.

D.1.1 Blender in general

Blender is a powerful, open-source 3D creation software that supports the full pipeline of 3D content production, including modeling, rigging, animation, simulation, rendering, compositing, and video editing. It features a physically based rendering engine advanced geometry manipulation tools, and support for scripting through a Python API (`bpy`).

bpy Through `bpy`, one can either procedurally generate 3D content (e.g., meshes, lights, cameras) directly in Python or import pre-designed assets from `.blend` files, such as those downloaded from online asset libraries. The core data structure of Blender is organized around key modules such as `bpy.data` (for accessing all objects), `bpy.context` (for active elements), and `bpy.ops` (for operator-based actions, such as `bpy.ops.object`, `bpy.ops.mesh`, etc.). Scenes can be configured via `scene.render` settings (e.g., resolution, output format), while materials and shaders are managed using node trees (`material.node_tree.nodes` and `...links`). Objects can be manipulated via their transforms (`location`, `rotation_euler`, `scale`), and geometries can be edited through mesh operations. This modular and scriptable architecture allows for flexible automation of complex 3D workflows, enabling precise control over content generation.

bpy.data The `bpy.data` module provides access to all data blocks in a Blender project, including objects, meshes, materials, cameras, lights, scenes, and more. These data blocks represent the raw components of a 3D scene and are not bound to any particular context. For example, `bpy.data.objects` retrieves the list of all objects in the project, while `bpy.data.materials` provides access to defined materials. This module is essential for querying, modifying, or creating persistent elements in a scene and is typically used when full control over the data structure is needed, independent of what is currently selected or active in the user interface.

bpy.context The `bpy.context` module refers to the current state of the Blender interface and holds references to the active scene, selected objects, active camera, and more. Unlike `bpy.data`, which is global, `bpy.context` is transient and dynamic—useful when writing scripts that depend on user selection or UI actions. For example, `bpy.context.active_object` accesses the object currently selected, allowing for context-sensitive manipulation such as transforming geometry or editing meshes in the right mode. Many Blender operators rely on context to function properly.

bpy.ops The `bpy.ops` module provides access to Blender's built-in operators, which are high-level actions that simulate interactive operations performed in the UI. This includes functionality for object manipulation (`bpy.ops.object`), mesh editing (`bpy.ops.mesh`), rendering, importing/exporting,

and more. For example, `bpy.ops.mesh.primitive_cube_add()` creates a new cube in the scene. Operators are context-sensitive and often require that the relevant object or mode be active.

scene.render The `scene.render` settings control output and rendering configurations for a Blender scene. This includes image resolution, file format, frame rate, output file paths, and rendering engine parameters. For example, `scene.render.resolution_x` and `scene.render.image_settings.file_format` are used to specify the dimensions and format (e.g., PNG, JPEG) of rendered images.

Materials and Node Trees Materials in Blender define the visual appearance of surfaces and can range from simple color definitions to complex shader graphs. These materials are composed using node trees, accessible via `material.node_tree.nodes` and `material.node_tree.links`. Nodes define operations (e.g., shaders, textures, math functions), while links connect their inputs and outputs to form a graph that defines the material behavior. Materials can be assigned to objects via their data (e.g., `object.data.materials`) and dynamically created or modified via Python, enabling procedural control over appearance.

Object Transformation and Manipulation Individual objects in Blender can be manipulated via their transformation attributes: `location`, `rotation_euler`, and `scale`. These properties are vectors that define an object's position, orientation, and size within the scene. For instance, setting `obj.location = (1.0, 0.0, 0.0)` translates the object along the x-axis. Objects also have a `data` field that links them to their geometric representation (e.g., mesh or camera) and can be duplicated, hidden, or animated.

Mesh Editing Meshes are editable geometric structures consisting of vertices, edges, and faces. Mesh editing can be performed using both `bpy.ops.mesh` operators (e.g., extrude, subdivide) and direct manipulation via the mesh data structure (e.g., `obj.data.vertices`). The former is suitable for higher-level transformations, while the latter allows low-level procedural generation of geometry. Switching to edit mode and updating the mesh with `bpy.ops.object.mode_set(mode='EDIT')` is often required for certain operations.

D.1.2 Dataset General Setup

Overview The dataset generation process is built on a modular and extensible framework that separates configuration, construction, and orchestration of all scene elements. This structure allows precise control, high reproducibility, and straightforward extensibility for synthetic data production.

Models Layer All scene parameters are structured as strongly-typed data classes (models) defined in `models.py` files. Each model captures the configuration and validation logic for a specific scene component.

- **MaterialModel**: Defines appearance properties for surfaces.
  `color` Default: (0.8, 0.8, 0.8, 1.0). RGBA tuple or string (e.g., `"white"`).
  `roughness` Default: 0.5. Controls micro-surface scattering (0.0 = mirror, 1.0 = diffuse).
  `material_name` Default: None. Optional string identifier for reuse or tracking.
  `custom_material` Default: None. Optional Blender reference.
- **CameraModel**: Controls camera position, orientation, and variability.
  `distance` Default: `"medium"`. Preset or float; distance from scene center.
  `angle` Default: `"medium"`. Preset or float; vertical angle (90° = top-down).
  `horizontal_angle` Default: 0.0. Azimuthal rotation (0–360°).
  `randomize_distance` Default: False. Toggles distance randomization.
  `randomize_distance_percentage` Default: 0.1. Range for distance variation.
  `randomize_angle` Default: False. Toggles angle randomization.
  `randomize_angle_percentage` Default: 0.1. Range for angle variation.
- **TableModel**: Configures the table supporting objects in the scene.
  `shape` Default: RECTANGULAR. Options: RECTANGULAR, CIRCULAR, ELLIPTIC.

`length` Default: 2.0. Length along x-axis (in Blender units).
`width` Default: 1.0. Width along y-axis.
`height` Default: 0.9. Table height.
`texture` Default: WOOD. Options: WOOD, MARBLE, METAL.
`material` Nested `MaterialModel`. Defines surface appearance.

- **FloorModel**: Configures the ground surface below the table.

  `color` Default: (0.8, 0.8, 0.8, 1.0). Floor base color.
  `roughness` Default: 0.5. Controls reflection properties.
  `material` Nested `MaterialModel`. Full material definition.
- **BackgroundModel**: Controls the visual and lighting background of the scene.

  `color` Default: (0.5, 0.5, 0.5, 1.0). Used when HDRI is disabled.
  `use_hdri` Default: False. Enables HDRI-based lighting.
  `hdri_path` Default: None. File path to HDRI texture (if enabled).
- **LightingModel**: Configures directional and ambient scene lighting.

  `lighting` Default: `"medium"`. Presets: very_low (0.3) to very_high (2.0), or float.
  `key_light_power` Default: 300.0. Primary light intensity.
  `fill_light_power` Default: 50.0. Fills shadows.
  `back_light_power` Default: 50.0. Enhances object separation.
- **ResolutionModel**: Defines the output image resolution and quality.

  `width` Default: 1920. Output width (pixels).
  `height` Default: 1080. Output height (pixels).
  `resolution_percentage` Default: 100. Renders at percentage of full resolution.
  `pixel_aspect_x`, `pixel_aspect_y` Default: 1.0. Pixel shape scaling.
  `randomize` Default: False. Toggles resolution variation.
  `randomize_percentage` Default: 0.1. Range for randomized resolution.

  Presets values:
  - `"low"`: 640×480
  - `"medium"`: 1280×720
  - `"high"`: 1920×1080
- **RenderModel**: Specifies render engine and output parameters.

  `engine` Default: `"CYCLES"`. Options: CYCLES, EEVEE, WORKBENCH.
  `samples` Default: 128. Number of render samples (higher = better quality).
  `exposure` Default: 0.0. Global brightness adjustment (in stops).
  `file_format` Default: `"PNG"`. Options: PNG, JPEG, TIFF, EXR, etc.
  `resolution` Nested `ResolutionModel`. Render size configuration.
  `output_path` Default: None. Output directory path.
  `gpu_enabled` Default: True. Enables GPU rendering when available.
  `gpus` Default: None. GPU device indices (e.g., [0, 1]).
- **SceneSetupModel**: Aggregates all sub-models into a single, hierarchical configuration.

Builder Layer (Scene Construction Logic) The construction of the scene is managed by a coordinated set of builder modules:

- `general_setup.py`: Orchestrates the end-to-end construction pipeline, calling individual builders in the correct order (scene clearing, resolution, render setup, environment, table, floor, camera, lighting).
- `camera.py`: Implements camera placement logic, including support for controlled randomization.
- `rendering.py`: Handles render engine configuration and scene cleanup.
- `resolution.py`: Defines and applies resolution presets or explicit overrides.
- Additional modules manage table, floor, background, and lighting setup.

Integration Layer (Pipeline Usage) The configuration for a scene is typically defined as a JSON or YAML dictionary, parsed and passed to the scene setup functions. The integration layer ensures:

- Loading and validating configuration files.
- Sequential invocation of all builder modules to assemble the scene.
- Full serialization/deserialization for reproducibility.
- Support for parameter overrides and randomization (for experiment diversity).

An example of configuration is given below:

```
config = {
    "resolution": {"resolution": "high"},
    "camera": {
        "distance": "medium",
        "angle": 60.0,
        "randomize_angle": True,
        "randomize_angle_percentage": 0.1
    },
    "table": {
        "shape": "rectangular",
        "length": 2.0,
        "width": 1.0,
        "material": {"color": (0.8, 0.7, 0.6, 1.0)}
    },
    "lighting": {"lighting": "high"}
}
```

D.1.3 Dataset Noise

Overview The `noise` module introduces systematic, configurable perturbations into the rendering pipeline to enhance dataset diversity and realism. It focuses on three main types of noise: (i) *blur noise*, simulating camera depth-of-field effects; (ii) *light noise*, introducing controlled variations in lighting intensity and direction; and (iii) *table texture noise*, varying the complexity of the material applied to the table surface.

Models Layer (`noise/models.py`) The configuration is built upon a set of strongly-typed data-classes defined in `models.py`. A common abstract base, `BaseNoiseModel`, defines shared attributes across noise types, and specific subclasses extend this base:

- **BaseNoiseModel**: Abstract base class for all noise types.

  `enabled` Default: `True`. Toggles whether the noise effect is applied.

  `intensity` Default: `1.0`. Float in range 0.0–2.0, defines effect strength.

  `seed` Default: `None`. Optional integer seed for reproducibility.

  `blend_mode` Default: `"MIX"`. Blend type: `"MIX"`, `"ADD"`, `"MULTIPLY"`, etc.

  `opacity` Default: `1.0`. Float in range 0.0–1.0, determines visual strength of the effect.

- **BlurNoiseModel**: Simulates depth-of-field blur by manipulating camera aperture (f-stop) settings.

  `intensity` Default: `"none"`. Accepts preset or float value.

  Presets:
  - `"none"`: No blur (disabled)
  - `"very_low"`: f/9.0
  - `"low"`: f/4.0
  - `"medium"`: f/2.0
  - `"high"`: f/1.0
  - `"very_high"`: f/0.5

- **LightNoiseModel**: Applies global lighting variation while maintaining a balanced three-point setup.

  `lighting` Default: `"medium"`. Controls global brightness via scalar presets.

  Presets:

- `"very_low"`: 0.3×
- `"low"`: 0.6×
- `"medium"`: 1.0× (default)
- `"high"`: 1.5×
- `"very_high"`: 2.0×

Base values (medium):
- Key Light: 400
- Fill Light: 200
- Back Light: 300

- **TableTextureNoiseModel**: Adjusts the complexity of table surface textures using procedural shaders.

  `table_texture` Default: `"medium"`. Controls texture entropy.

  Presets:
  - `"low"`: Monochrome surface (simple color + roughness)
  - `"medium"`: Color noise and variation
  - `"high"`: Multi-layer textures with bump mapping

  Color Palettes (low entropy):
  - Light gray: (0.8, 0.8, 0.8, 1.0)
  - Medium gray: (0.6, 0.6, 0.6, 1.0)
  - Dark gray: (0.4, 0.4, 0.4, 1.0)
  - Light wood: (0.8, 0.7, 0.6, 1.0)
  - Medium wood: (0.6, 0.5, 0.4, 1.0)
  - Dark wood: (0.4, 0.3, 0.2, 1.0)

- **NoiseConfigModel**: Serves as the master container for all noise-related configuration models.

  `blur` Optional `BlurNoiseModel` instance.

  `light` Optional `LightNoiseModel` instance.

  `table_texture` Optional `TableTextureNoiseModel` instance.

Builder and Implementation Layer Each noise type has a dedicated implementation module responsible for its operational logic:

- `blur.py`: Sets depth-of-field parameters on the active camera. F-stop values define the blur intensity, and focus distance is automatically computed when not explicitly set.
- `light.py`: Applies a standardized three-point lighting configuration, scaling intensity uniformly across presets and maintaining key/fill/back ratios.
- `table_texture.py`: Generates shader node trees with increasing entropy levels, from uniform color to layered procedural patterns and bump maps.
- Additional modules handle distractors or future noise types.

Integration Layer The module `set_noise_config.py` orchestrates the application of noise by parsing the unified configuration, instantiating relevant model classes, delegating to builder functions, and applying the results directly to the scene. The integration returns a composite summary of the applied noise elements and parameters for inspection or reuse.

Configuration System The noise system supports both human-readable and programmatic usage. Named presets such as `"low"`, `"medium"`, and `"high"` map to technical parameters (e.g., f-stop, light multiplier, entropy level). All configuration models provide explicit defaults and can be serialized into JSON or YAML.

An example of configuration is given below:

```
noise_config = {
    "blur": "medium", # 2.0 f-stop
    "lighting": "high", # 150% intensity
    "table_texture": "low" # Simple monochrome surface
}
```

D.1.4 Chess Base Content

Overview The chess module provides specialized components for generating parametrizable chess boards and pieces in Blender. It supports both procedural and asset-based (e.g., `.blend` file) geometry and material assignment, with a focus on reproducibility, modularity, and flexibility for synthetic vision tasks.

Models Layer (`chess/models.py`) The chess dataset generation is structured around two primary models: the **BoardModel** and the **PieceModel**, both implemented as dataclasses.

- **BoardModel**: Configures the geometry, layout, and appearance of the chess board.
  `length` Default: 0.7. Board length.
  `width` Default: 0.7. Board width.
  `thickness` Default: 0.05. Thickness of the board.
  `border_width` Default: 0.05. Width of the board border.
  `location` Default: (0, 0, 0.9). Board position in world coordinates.
  `rows` Default: 8. Number of rows (must be a power of 2).
  `columns` Default: 8. Number of columns (must be a power of 2).
  `random_pattern` Default: False. Enables random square pattern.
  `pattern_seed` Default: None. Seed for deterministic pattern generation.
  `board_material` Nested `MaterialModel`. Material for board frame.
  `white_material` Nested `MaterialModel`. Material for white squares.
  `black_material` Nested `MaterialModel`. Material for black squares.
- **PieceModel**: Configures geometry, placement, and material for a chess piece.
  `piece_type` Type of piece (e.g., king, queen, rook, etc.).
  `location` Default: (0, 0, 0). World-space placement.
  `material` Nested `MaterialModel`. Controls appearance.
  `geometry` Nested `GeometryModel`. Scale, rotation, randomness.
- **GeometryModel** (for pieces):
  `scale` Default: 0.1. Piece size scaling.
  `random_rotation` Default: False. Enables random rotation.
  `max_rotation_angle` Default: 15.0. Maximum rotation angle (degrees).
  `seed` Optional integer for reproducibility.
- **MaterialModel** (shared, see previous section): Color, roughness, material name/reference, and optional custom Blender material.

Board Builder Layer

- Implemented in `board.py`. Handles geometry generation, procedural or custom material assignment, pattern creation (checkerboard or random), and spatial tracking of all cell positions.
- Board is built in layers: frame $\rightarrow$ squares $\rightarrow$ collection assignment.
- Exposes methods to query or iterate over cell positions for piece placement.
- Includes validation (power-of-2 grid) and scene cleanup.

Piece Builder Layer

- Implemented in `factories.py`. Pieces are instantiated from configurations, either procedurally or by extracting meshes from `.blend` files.
- Abstract base factory, with implementations for:
  - **DefaultPieceFactory**: Procedural.
  - **OldSchoolPieceFactory**: Imports from `.blend` assets.
  - **StonesColorPieceFactory**: Imports from `.blend` assets.
- Piece classes encapsulate geometry creation, positioning, material assignment, and (if needed) mesh caching for performance.

Integration Layer The chess board and pieces are integrated within the main dataset generation scripts (e.g., `generate_chess_image.py`):

- Instantiate `ChessBoard` with configuration, build board, track cell positions.
- For each piece, use cell position and piece configuration to instantiate and place with the chosen factory.
- Supports switching styles by factory selection, and parameterization by passing different configs.

An example of board configuration is given below:

```
board_config = {
    "length": 0.7,
    "width": 0.7,
    "thickness": 0.05,
    "location": (0, 0, 0.9),
    "border_width": 0.05,
    "rows": 8,
    "columns": 8,
    "random_pattern": False,
    "board_material": {
        "color": (0.4, 0.3, 0.2, 1.0),
        "roughness": 0.5
    },
    "white_material": {
        "color": (0.9, 0.9, 0.9, 1.0),
        "roughness": 0.3
    },
    "black_material": {
        "color": (0.1, 0.1, 0.1, 1.0),
        "roughness": 0.3
    }
}
```

An example of piece configuration is given below:

```
piece_config = {
    "type": "king",
    "location": (0, 4),
    "color": (0.9, 0.9, 0.9, 1.0),
    "scale": 0.08,
    "random_rotation": True,
    "max_rotation_angle": 10.0,
    "roughness": 0.3,
    "material_name": "KingMaterial"
}
```

D.1.5 Chess Generation Logic

Overview The chess dataset generation system orchestrates the production of diverse and configurable chess scene images. It is built around a layered, modular architecture supporting both high-level experiment definition and low-level control over piece, board, and noise configurations.

Core Components

- **Dataset Configuration**: High-level `yaml` files (`dataset_configs/*.yml`) define global experiment parameters and variable selection strategies.
- **Variable Management**: `generate_dataset.py` processes variable definitions and manages generation of configuration combinations.
- **Advanced Models**: `advanced_models.py` includes data models for piece count, type, and position, enabling expressive and precise dataset definition.
- **Configuration Generation**: `advanced_generator.py` provides generators to build concrete board and piece configurations from high-level specs.
- **Image Generation**: `generate_img_from_yaml.py` instantiates configurations, selects piece style factories, and coordinates the rendering pipeline.

Dataset Configurations `yaml` files specify the experimental setup, with explicit variable control and experiment metadata. For example:

```yaml
dataset:
  name: chess_identification
  output_dir: ../data/chess_ident_dataset
  seed: 42
  piece_set: old_school

variables:
  chess.count_config:
    variate_type: fixed
    variate_levels:
      type: fixed
      value: 3
      randomization: false

  chess.type_config:
    variate_type: varying_random
    variate_levels: [pawn, rook, knight, bishop, queen, king]
    n_images: 10

  noise.blur:
    variate_type: varying_all
    variate_levels: [none, very_low, low, medium,
                     high, very_high]
    n_images: 5
```

Variable types supported:

- Fixed (`fixed`): Single constant value
- All possible values (`varying_all`): Test all enumerated values
- Random selection (`varying_random`): Sample randomly from a list
- Range (`varying_among_range`): Sample from a continuous range

Variable Management Variable handling is managed by data classes and utility classes:

```python
@dataclass
class VariableConfig:
    variate_type: str
    variate_levels: Union[Any, List[Any], Tuple[Any, Any]]
    n_images: int = 1
    randomize: bool = False
    randomize_percentage: float = 0.2
```

Core utility classes:

- **VariableCombinationGenerator**: Builds all combinations of configuration parameters from high-level experiment definitions.
- **DatasetConfig** and **ConfigConverter**: Aggregate and convert variables into structured configurations for downstream generators.

Advanced Models (`advanced_models.py`) **PieceCountModel**: Specifies how many pieces to place on the board.

```python
@dataclass
class PieceCountModel:
    spec_type: CountSpecificationType = \
               CountSpecificationType.PRESET
    preset: str = "medium"
    count: int = 10
    min_count: Union[int, str] = 5
    max_count: Union[int, str] = 15
```

PieceTypeModel: Specifies which piece types to use.

```
@dataclass
class PieceTypeModel:
    spec_type: PieceTypeSpecification = \
               PieceTypeSpecification.PRESET
    preset: str = "medium"
    types: List[str] = field(default_factory=list)
    n_types: int = 3
```

PiecePosition: Controls spatial distribution and placement constraints.

```
@dataclass
class PiecePosition:
    allowed_positions: List = field(default_factory=list)
    spread_level: SpreadLevel = SpreadLevel.MEDIUM
    start_point: Union[StartingPoint, Tuple] = \
                            StartingPoint.CENTER
```

Configuration Generation (`advanced_generator.py`) Generators for each major component:

- **PieceCountGenerator**, **PieceTypeGenerator**, **PiecePositionGenerator**: Convert high-level specs into detailed configurations.
- **ChessConfigGenerator**: Orchestrates board and piece config creation.

Image Generation (`generate_img_from_yaml.py`) The `ChessImageGenerator` class connects configuration and rendering:

```
class ChessImageGenerator:
    def __init__(self, config_path=None,
                 config_dict=None, piece_set=None):
        # Initialize with config source...

    def load_config(self):
        # Load and validate configuration...

    def _create_chess_config(self):
        # Process chess-specific configuration...

    def generate_image(self, output_dir=None,
                       base_filename="chess_scene"):
        # Generate the chess image...
```

Configuration and Execution Flow

1. **Define dataset configuration** (YAML): Select variables, types, values, and randomization.
2. **Generate variable combinations**: VariableCombinationGenerator produces exhaustive or sampled experimental setups.
3. **Build chess configuration**: ChessConfigGenerator converts high-level variables into board and pieces configs.
4. **Generate images**: ChessImageGenerator runs the rendering, using selected style factories and noise configs.
5. **Trace metadata**: Outputs both images and accompanying legend files for reproducibility.

Example Application Scenarios

- **Piece Identification Tasks**: Varying piece types with controlled counts

  ```
  chess.type_config:
    variate_type: varying_all
    variate_levels: [pawn, rook, knight, bishop, queen, king]
  ```

- **Position Localization Tasks**: Distributing pieces with different spread patterns

```
chess.position_config.spread_level:
  variate_type: varying_all
  variate_levels: [low, medium, high]
```

- **Visual Robustness Testing**: Adding controlled noise to images

```
noise.blur:
  variate_type: varying_all
  variate_levels: [none, low, medium, high]
```

- **Count Estimation Tasks**: Varying the number of pieces

```
chess.count_config:
  variate_type: varying_among_range
  variate_levels: [5, 25]
```

This sophisticated architecture enables researchers to create precisely controlled, reproducible, and diverse chess image datasets for training and evaluating computer vision models on tasks like object identification, counting, and localization.

D.1.6 Poker Generation Logic

Overview The poker dataset generation framework closely mirrors the modular architecture of the chess system. It consists of layered components for configuration, variable management, and scene construction. However, the poker framework introduces domain-specific logic for player management, river (community cards) handling, and flexible card/chip distribution.

Core Architecture

- **Dataset Configuration Layer**: High-level YAML files (`dataset_configs/*.yml`) define global parameters and variable strategies.
- **Variable Management Layer**: (`generate_dataset.py`) Generates all combinations of experimental variables, tracking metadata for reproducibility.
- **Scene Generation Layer**: (`scene_generator.py`) Systematically builds up poker scenes according to input configuration.
- **Component Building Layer**: Modular builders for cards, players, chip stacks, and community (river) cards.

Configuration System As with chess, the configuration is driven by YAML files specifying the dataset parameters and experimental variables. For example:

```
dataset:
  name: poker_chip_variations
  output_dir: /home/...
  seed: 5678

variables:
  n_players:
    variate_type: fixed
    variate_levels: 4

  card_distribution_inputs.overall_cards:
    variate_type: varying_all_range
    variate_levels: [2, 15]
    n_images: 2

  chip_distribution_inputs.color_options:
    variate_type: varying_among
    variate_levels: [ [1.0, 0.0, 0.0, 1.0], [0.0, 0.0, 1.0, 1.0]]
```

The system supports variable types such as `fixed`, `varying_all`, `varying_random`, and `varying_all_range`, for both card and chip configurations, as well as camera and noise parameters.

Poker-Specific Logic While the general workflow is analogous to chess, poker dataset generation introduces:

- **Player Logic**: Each player receives a configurable hand of cards and chip stacks, with control over card selection, arrangement, face-up/face-down mix, and chip pile details.
- **River (Community) Logic**: The framework allows specification of board/river cards, including precise arrangement, card identities, and spread.
- **Card Distribution Logic**: Flexible mechanisms for dealing cards to players and the river, including control over overlap, layout (horizontal, vertical, auto), and visual spread.

Example of player and river configuration:

```
{
  'player_id': 'Alice',
  'hand_config': {
      'card_names': ['AS', 'AH', 'AD', 'AC', '2S'],
      'n_cards': 5,
      'location': (-0.6, 0.0, table_height + 0.01),
      'scale': 0.1,
      'spread_factor_h': 0.2,
      'spread_factor_v': 0.05,
      'n_verso': 0,
      'random_seed': 101
  },
  'chip_area_config': {
      "base_pile_config": {
          "n_chips": 8,
          "base_chip_config": {"chip_object_name": "Cylinder001",
                               "scale": 0.06,
                               "color": (0.1, 0.2, 0.8, 1)},
          "spread_factor": 0.1
      },
      "n_piles": 2,
      "n_chips_per_pile": [8, 10],
      "pile_colors": [None, (0.2, 0.8, 0.2, 1)],
      "pile_spreads": [None, 0.3],
      "random_seed": 1001
  }
}

"community_cards": {
    'card_names': ['4C', '4H', '4D', '4S', '5C'],
    'n_cards': 5,
    'start_location': (-0.3, 0, 0.9 + 0.01),
    'scale': 0.1,
    'n_verso': 0,
    'card_gap': {'base_gap_x': 0.15, 'base_gap_y': 0.005,
                 'random_gap': False}
}
```

Key Features and Variations

- **Flexible Player Setup**: Any number of players, each with distinct hands and chip stacks.
- **Customizable Card Layouts**: Support for horizontal, vertical, or auto layout, and explicit overlap/spread control.
- **Community/River Cards**: Positioning and spread of board cards, face-up or face-down.
- **Chips**: Color, pile count, and scale randomized or specified per player.
- **Noise, Camera, and Lighting**: Full control over noise (e.g., blur), camera angles, and lighting as in chess.

Summary In summary, the poker dataset generation logic generalizes the chess framework to card-based games, with additional modules for modeling players, community cards, and complex card/chip arrangements. The design enables researchers to create richly annotated, reproducible datasets for AI tasks such as card identification, counting, and arrangement analysis.

D.2 Questions

The question system provides a structured, extensible framework for generating and formatting questions to assess visual reasoning abilities on chess and poker scenes. This system supports a wide spectrum of cognitive tasks, question formulations, preprompt strategies, and instruction combinations.

D.2.1 Main content

Questions are defined per domain in a central `all_questions.json` file, covering both chess and poker. Each game includes a rich variety of question types, covering counting, identification, localization, and compositional reasoning.

Categories and Examples

- **Counting**:
  `count_pieces` — "How many pieces are there in the image?"
  `count_total_cards` — "How many cards are present in the entire scene (including all hands and community cards)?"
- **Identification**:
  `identify_type_one_piece` — "What piece is on the board?"
  `identify_cards` — "What are the cards on the table?"
- **Localization**:
  `localize_column_one_piece` — "On which column is the piece on the board?"
  `count_community_cards` — "How many community cards are visible on the table?"
- **Combined/Complex**:
  `count_identification_white_pieces` — "How many white pieces are there on the board?"
  `count_identify_face_up_cards` — "How many cards are face up on the table?"

Questions are selected and instantiated by key (e.g., `count_pieces`), with category and format determined by the handler logic.

D.2.2 preprompts

preprompts are optional instructional phrases prepended to the main question to guide model reasoning or control for dataset biases. Supported preprompts include:

- **Debiased:**
  Chess: "This is not a real chess game. The number of each piece and their position can vary arbitrarily. Just focus on answering the following question based on the visual content."
  Poker: "This is not a real poker game. The number of each card and their position can vary arbitrarily. Just focus on answering the following question based on the visual content."
- **Chain-of-Thought (CoT):**
  "First, think carefully, step by step, about the question being asked and the relevant elements of the image to which the question refers. End your message with {answer : <answer>}"
- **Debiased CoT:**
  Combines both debiased and chain-of-thought instructions.

preprompts can be flexibly combined with any question or instruction format. For example:

> *Debiased CoT for chess*:
> "This is not a real chess game. The number of each piece and their position can vary arbitrarily. Just focus on answering the following question based on the visual content. First, think carefully, step by step, about the question being asked and the relevant elements of the image to which the question refers. End your message with {answer : <answer>}"

D.2.3 Instructions

We can automatically add to the main question instructions to obtain various answer styles. The system provides:

- **Direct question** (default):
  "How many pieces are there in the image?"
- **Declarative statement**:
  "The number of pieces in the image is:"
- **Fill-in-the-blank**:
  "There are ____ pieces in the image."

D.2.4 Overall (combinations)

The system enables systematic generation of question variants for robust evaluation. For each question, the handler can:

- Select domain (chess or poker)
- Specify question key(s) or task category
- Choose instruction format (direct, declarative, fill-in-blank)
- Add any preprompt(s) (debiased, CoT, debiased_cot)
- Optionally substitute variables (e.g., position X,Y) from image metadata

Example Combinations

- **Basic, direct:**
  "How many pieces are there in the image?"
- **With CoT preprompt:**
  "First, think carefully, step by step, about the question being asked and the relevant elements of the image to which the question refers. How many pieces are there in the image?"
- **Declarative, with debiased prompt:**
  "This is not a real poker game. The number of each card and their position can vary arbitrarily. Just focus on answering the following question based on the visual content. The number of cards present in the entire scene (including all hands and community cards) is:"
- **Fill-in-blank with CoT:**
  "First, think carefully, step by step, about the question being asked and the relevant elements of the image to which the question refers. There are ____ cards in the entire scene (including all hands and community cards)."

D.3 Legend and answers

D.3.1 Legends

The legend generation system documents key metadata for each generated image, supporting traceability, dataset evaluation, and model training. Chess and poker domains use parallel architectures, adapted for their respective scene elements.

Process Overview

1. **Convert Configurations**: Parse internal configuration dictionaries (scene, board/table, pieces/-cards, camera, noise) into a unified intermediate structure.
2. **Build Hierarchical Legends**: Construct structured dictionaries capturing all relevant parameters and object details.
3. **Format Legends**: Output both human-readable text files and machine-readable JSON files.
4. **Write to Disk**: Save legends alongside rendered images for downstream use.

Chess Legends

- **Board**: Dimensions, pattern, position, material, and colors
- **Pieces**: Type, position (board/world), color, scale, and other properties
- **Camera**: Distance, angles, world coordinates
- **Noise**: Blur, lighting, table texture

Poker Legends

- **Scene Setup**: Table, camera, lighting, render parameters
- **Cards**: Player hands (cards, position, face-up/down), community cards, layout
- **Players**: IDs, hand details, chip stacks
- **Noise**: Visual effects applied

Example Output (Excerpt)

```
CHESS PIECES LEGEND
KING PIECES (2):
- KING_1: Board Position: row 0, col 4; Color: RGBA(0.9, 0.9, 0.9, 1.0); Scale:
0.08

COMMUNITY CARDS (POKER):
Cards: 7D, 3C, 9H
Player: Player_1; Hand Cards: JC, KS
```

Key Features

- **Dual Output**: Text for readability, JSON for automation
- **Comprehensive**: Covers all scene parameters and objects
- **Domain-Specific**: Adapted for both chess and poker elements
- **Error Robust**: Handles missing or invalid data gracefully

This system ensures every image is accompanied by a complete, interpretable record, supporting reliable evaluation of visual reasoning tasks.

D.3.2 Answer extraction

The `AnswerExtractor` class implements a principled, automated approach for extracting ground-truth answers from structured image legends. This process is fundamental for evaluating visual question answering (VQA) systems, as it guarantees alignment between the generated images, the underlying parameters, and the target answers used for benchmarking.

Core Extraction Mechanism. At its core, the extractor operates on JSON-formatted legends generated during image synthesis. The main extraction method is driven by the *question key* and *game type*, optionally leveraging the full question text for context. Upon receiving a legend dictionary, the extractor first routes the request according to the question type, applies any necessary context-aware logic (e.g., parsing the question to identify parameters like piece or suit), and formats the output in the appropriate structure (string, list, or dictionary). This systematic approach supports a wide range of question and answer formats.

Domain-Specific Logic: Chess. For chess images, the extractor covers a variety of tasks, including:

- **Simple Counting:** Returns the number of pieces present.
- **Board Properties:** Computes the number of squares from board dimensions.
- **Piece Localization:** Retrieves the row or position of a given piece.

- **Piece Identification:** Identifies the type of a single piece.
- **Spatial Reasoning:** Calculates the spatial distance (e.g., rows apart) between pieces.
- **Color-Based Questions:** Counts the number of pieces by color (e.g., white pieces).

Each question type is mapped to dedicated extraction logic, ensuring accurate and reliable retrieval from nested legend structures.

Domain-Specific Logic: Poker. For poker images, the extraction logic is adapted to card game semantics and includes:

- **Card Counting:** Aggregates the total number of cards by summing community, player, and overlap cards.
- **Card Identification:** Lists the community cards or retrieves specific cards shown.
- **Player-Based Questions:** Determines which player has the most cards in hand.
- **Attribute-Based Questions:** Parses the question text (e.g., via regex) to count cards matching a target suit or attribute.

This flexible, parameter-driven logic ensures coverage of the broad spectrum of poker VQA tasks.

D.4 Result retrieval

For each visual reasoning task, model predictions are retrieved using a dedicated function that interfaces with the selected Vision Language Model (VLM) provider. For every image-question pair, the system submits both the image and the formulated question to the VLM via an API call. This process is encapsulated in the central function `evaluate_VLM_on_task()`, which handles provider-specific formatting, request dispatch, and response parsing. The returned prediction may take various forms depending on the provider; hence, post-processing is applied to extract the actual answer text. For instance, if the response is a structured dictionary, the answer is extracted from the relevant field. In cases where numeric values are expected, regular expressions are used to robustly extract numerical information from free-form model outputs. All predictions are collected and stored for downstream evaluation, including accuracy calculation and error analysis.

E Correlation between synthetic and real data

To assess the robustness and real-world applicability of Vision-Language Models (VLMs) on visual reasoning tasks, we evaluated the correlation between their predictions on synthetic chess images and on matched real-world reproductions. Each model was tasked with counting chess pieces ranging from 1 to 8, with synthetic and real images covering the same set of target counts. For each number of pieces, we reproduce in real life 10 chess scenes taken from the synthetic data, using their legend as ground truth (see Figure 50).

Figure 50: Examples of synthetic images (a) and their real reproductions (b).

As shown in Figures 515253, most models achieve high alignment between synthetic and real predictions. When we average over all ten images per level and all preprompt and reformulation results, Pearson and Spearman correlation coefficients reach above 0.99 on the aggregated accuracies for the strongest models (GPT-4.1, GPT-4.1-mini, Mistral-small-3.1, and both LLaMA-4 models). However, performance drops are observed for smaller or less capable models, with correlations as low as 0.64 (Gemma3:12b), indicating substantial degradation in real-world generalization for these architectures. This high overall correlation for advanced models suggests that performance on synthetic diagnostic datasets is a strong proxy for real-world counting ability in structured visual tasks.

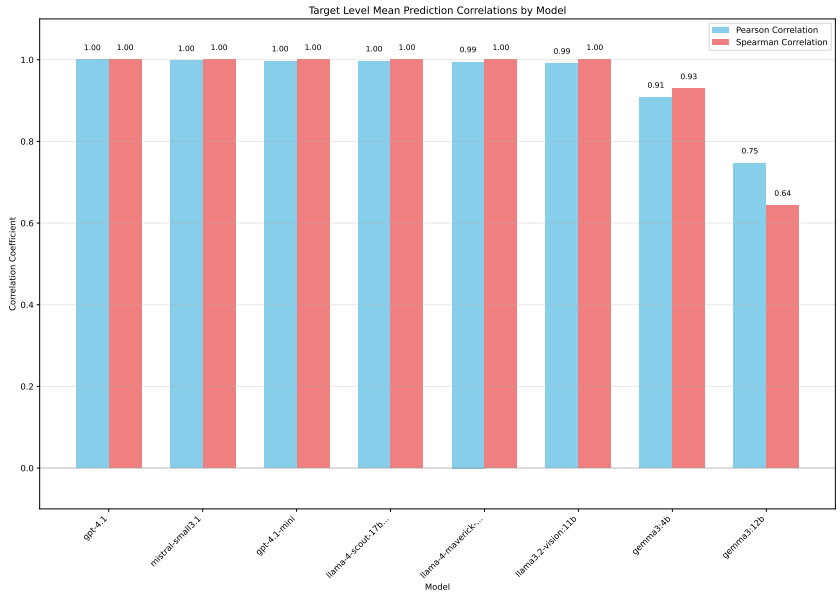


Figure 51: Correlation between model performance on synthetic and real images, aggregated by target level. For each number of pieces (1 to 8), we average the accuracy across all 10 samples and 6 preprompt-instruction pairs, then compute both Pearson and Spearman correlation coefficients over the 8 aggregated results.

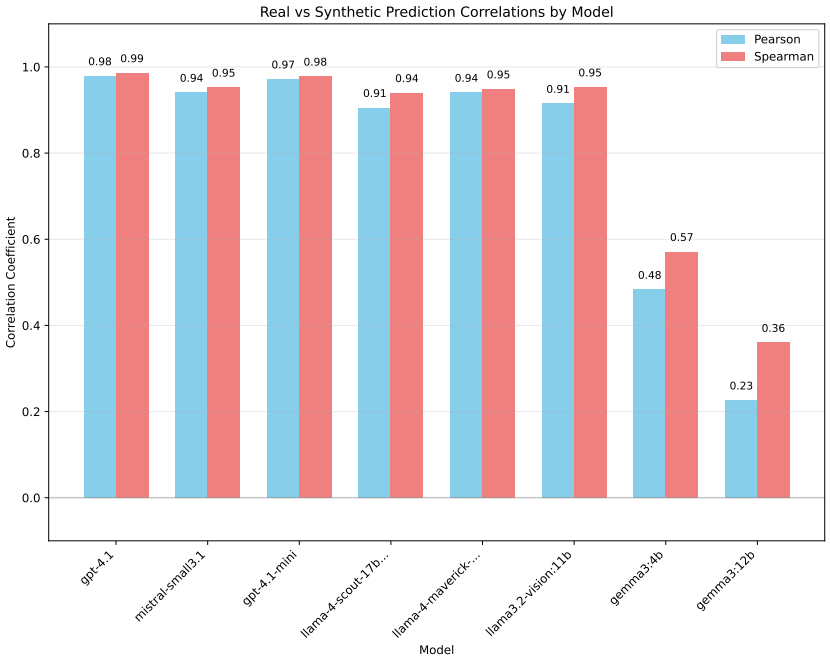


Figure 52: Correlation between model performance on synthetic and real images, computed at the individual prediction level. For each number of pieces (1 to 8), each preprompt-instruction pair, and each image scene (10 per level), we calculate both Pearson and Spearman correlation coefficients for accuracy. The average across all these scores is presented.

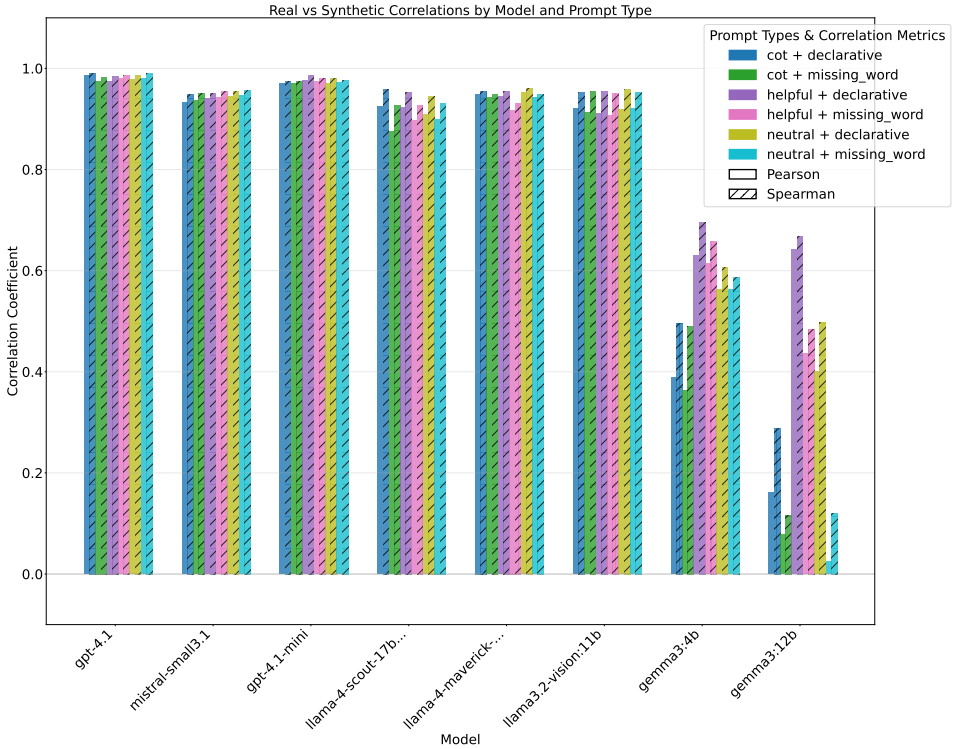


Figure 53: Correlation between model performance on synthetic and real images by instruction and preprompt type. For each number of pieces (1 to 8), we compute Pearson and Spearman correlation coefficients for accuracy across 10 images, then report the mean correlation for each prompt type.