**Structural Visual Intelligence**

by

Tiange Luo

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2026

Doctoral Committee:

Professor Honglak Lee, Co-Chair
Professor Justin Johnson, Co-Chair
Professor Joyce Y. Chai
Professor Todd Hollon

Tiange Luo

tiangel@umich.edu

ORCID iD: 0000-0001-9557-9817

© Tiange Luo 2026

# ACKNOWLEDGEMENTS

This dissertation would not have been possible without the guidance, support, and trust of my advisors, Honglak Lee and Justin Johnson. They gave me the freedom to explore broadly and helped shape the direction—always pushing me to clarify the core contributions and the evidence behind them. Honglak's emphasis on essential capabilities and foundational understanding, and Justin's insistence on simplicity, efficiency, and clarity, have profoundly shaped my research taste. Their mentorship has laid a foundation that I will carry with me long after this dissertation.

I am also grateful to my collaborators—Lajanugen Logeswaran, Chris Rockwell, Ruei-Che Chang, Harry Sanders, Ang Cao, Yunseok Jang, Yeda Song, Yiwei Lyu, Jaekyeom Kim, Jiacheng Shi, Yihong Tang, Ao Qu, Mohamed El Banani, and all my other collaborators—for their invaluable contributions, thoughtful discussions, and feedback that helped shape this work. I further appreciate the members of Honglak's and Justin's labs, as well as my colleagues at LG AI Research, for the many friendly interactions and shared activities that made this journey more enjoyable. I also thank the people in BBB; students in other AI labs; the CSE department staff; Brenda Imber, Judy Dyer, and the ELI department staff; Monica Roskamp and the International Center staff; and Aron Aliaga del Castillo and the UM dental school students and staff for their support and assistance throughout my time at the University of Michigan.

I also extend my sincere thanks to my committee members, Joyce Y. Chai and Todd Hollon, for their thoughtful feedback and encouragement throughout this journey.

Outside of research, I've been fortunate to find friends who made my PhD years meaningful and memorable. I'm especially grateful to Han Zheng for being there through countless ups and downs—gym-cooking sessions, travels, my defense, and so many moments I'll always remember. I

also thank Hanqi Zhu, Linyi Jin, Xili Yi, and Zihan Wu for organizing many fun activities. I am grateful to Zhengqi Xu, Xiao Li, Jing Peng, and Chris Rockwell for many great sports sessions. I thank Xiaolan Zhou for her companionship, which brightened my final year. I further thank Zixuan Huang, Wenzhe Tong, Jingyu Song, Zirui Xu, Yiqi Li, Ziyang Chen, Wenyi Wang, Fan Yang, Jiaming Yang, Ge Zhang, Qingyi Wang, Joyce Zhang, Hanqi Wen, and the many people I met during my PhD journey for shared meals, games, and countless moments of companionship.

Finally, I owe my deepest gratitude to my parents for their unconditional love and tremendous support throughout my growth, education, and career.

# TABLE OF CONTENTS

**ACKNOWLEDGEMENTS** . . . . . ii

**LIST OF FIGURES** . . . . . vii

**LIST OF TABLES** . . . . . xix

**LIST OF APPENDICES** . . . . . xxii

**ABSTRACT** . . . . . xxiii

**CHAPTER**

**1. Introduction** . . . . . 1

1.1 Thesis Overview . . . . . 3
1.2 First published appearances of the described contributions . . . . . 4

**2. Neural Shape Compiler: A Unified Framework for Transforming between Text, Point Cloud, and Program** . . . . . 6

2.1 Introduction . . . . . 6
2.2 Related Work . . . . . 9
2.3 Method . . . . . 11
2.3.1 Encoders & Decoders . . . . . 12
2.3.2 ShapeCode Transformer . . . . . 14
2.4 Experiments . . . . . 15
2.4.1 *Text* $\Longrightarrow$ *Point Cloud* . . . . . 16
2.4.2 *Point Cloud* $\Longrightarrow$ *Text* . . . . . 21
2.4.3 *Point Cloud* $\Longrightarrow$ *Program* . . . . . 23
2.4.4 *Partial Point Cloud* $\Longrightarrow$ *Complete Point Cloud* . . . . . 25
2.5 Limitations and Future Works . . . . . 26
2.6 Conclusion . . . . . 27

**3. Scalable 3D Captioning with Pretrained Models** . . . . . 29

3.1 Introduction . . . 30
3.2 Related Work . . . 32
3.3 Method . . . 34
3.3.1 Captioning Process . . . 34
3.3.2 Ethical Filtering . . . 36
3.4 Dataset . . . 37
3.4.1 Objaverse Captions . . . 37
3.4.2 ABO Geometry Captions . . . 38
3.5 Experiments . . . 39
3.5.1 3D Captioning on Objaverse . . . 40
3.5.2 Geometry 3D Captioning on ABO . . . 42
3.5.3 Large-Scale Text-to-3D Generation . . . 42
3.6 Limitations and Future Works . . . 45
3.7 Conclusion . . . 46

**4. View Selection for 3D Captioning via Diffusion Ranking** . . . 48

4.1 Introduction . . . 48
4.2 Related Work . . . 52
4.2.1 3D-Text . . . 52
4.2.2 Diffusion Model . . . 53
4.3 Method . . . 54
4.3.1 Issues in Cap3D . . . 54
4.3.2 DiffuRank Formulation . . . 55
4.3.3 New 3D Captioning Framework . . . 57
4.4 Dataset . . . 58
4.4.1 Correction of Cap3D Captions . . . 58
4.4.2 Dataset Expansion and Ethical Filtering . . . 59
4.5 Experiments . . . 60
4.5.1 Captioning Evaluation . . . 61
4.5.2 Text-to-3D Generation with New Captions . . . 63
4.5.3 DiffuRank on VQA . . . 64
4.6 Conclusion . . . 66

**5. Probing Language Priors on Visual Reasoning in VLMs** . . . 67

5.1 Introduction . . . 67
5.2 Related Work . . . 70
5.3 ViLP Benchmark . . . 72
5.3.1 Design Principles . . . 72
5.3.2 Question-Image-Answer Generation . . . 73
5.3.3 Dataset Evaluation . . . 75
5.4 Image DPO . . . 75
5.4.1 Objective . . . 75

5.4.2 Data Generation . . . 77
5.5 Experiments . . . 78
5.5.1 Image DPO . . . 83
5.6 Conclusion . . . 85

**6. Towards Minimal Fine-Tuning of VLMs** . . . 86

6.1 Introduction . . . 86
6.2 Related Works . . . 90
6.3 Method . . . 91
6.3.1 Background . . . 92
6.3.2 Image-LoRA . . . 93
6.3.3 Head Selection . . . 95
6.4 Experiments . . . 97
6.4.1 Image-LoRA vs. Standard LoRA . . . 97
6.4.2 Ablations on Image-LoRA . . . 102
6.5 Limitations and Discussions . . . 105
6.6 Conclusion . . . 106

**7. Visual Test-time Scaling for GUI Agent Grounding** . . . 107

7.1 Introduction . . . 107
7.2 Related Work . . . 110
7.2.1 GUI Agents . . . 110
7.2.2 Test-Time Scaling in AI Agents . . . 112
7.2.3 Visual Image Attention . . . 112
7.3 Method . . . 113
7.3.1 Overview . . . 113
7.3.2 Visual Region Focus . . . 114
7.3.3 Image-as-Map . . . 115
7.4 Experiments . . . 117
7.4.1 ScreenSpot-Pro . . . 117
7.4.2 WebVoyager . . . 120
7.4.3 Ablation Studies & Analysis . . . 121
7.5 Conclusion . . . 125

**8. Conclusion and Future Works** . . . 126

8.1 Conclusion . . . 126
8.2 Future Works . . . 127

**APPENDICES** . . . 130

**BIBLIOGRAPHY** . . . 264

# LIST OF FIGURES

**Figure**


2.1 Example: Neural Shape Compiler helps obtain hierarchical information for the reconstruction result of a single image (*Zhang et al.*, 2018), including its structural descriptions, regularities, and how to assemble it. . . . . 7

2.2 Overview, and three compilations conducted by Neural Shape Compiler, including *Text* ⇒ *Point Cloud*, *Point Cloud* ⇒ *Program*, and *Program* ⇒ *Text*Footnote 1. Red shapes are the rendered results by executing the corresponding program. ShapeCode Transformer is shared across different compilations. Neural Shape Compiler is not limited to performing the compilations shown above, and benefits from joint training with all heterogeneous data and tasks. . . . . 8

2.3 *Point*VQVAE consists of three parts: (1) a hierarchical encoder that has restricted receptive fields and outputs multiple part embeddings; (2) a shared codebook for vector quantization (*Van Den Oord et al.*, 2017) (e.g., looking up the closest yellow embedding in the codebook for the yellow output of the encoder); (3) a decoder with multiple residual building blocks consisting of 1x1 convolutional layers, a max-pooling layer, and MLP layers at the end to output 2,048 points. . 12

2.4 *Text* ⟹ *Point Cloud* by Neural Shape Compiler. A text prompt may translate to multiple shapes. . . . . 17

2.5 *Text* ⟹ *Point Cloud*. Neural Shape Compiler can generate corresponding point clouds to the text prompt with structural details, while the baselines generate inaccurate structures or fewer alignments with the text prompt. . . . . 18

2.6 Render the shape with six different camera positions and two different point lights by Pytorch3D *Ravi et al.* (2020). The texts shown at the top left are the shape-associated texts provided in our datasets, which only contain Round or Circular. We modified them into Rectangle or Rectangular as shown in the top right. The number above each image is the average cosine similarity between each CLIP image embedding and all corresponding CLIP text embedding. . . . . 20

2.7 *Point Cloud* ⟹ *Text*. One description per sentence. The shown shapes are from test sets, and Neural Shape Compiler tell their structures well. . . . . 21

2.8 *Point Cloud* ⟹ *Program*. Neural Shape Compiler can well infer programs for the shown shapes of unseen categories and reconstruct shapes in a creative manner (e.g., represent the semicircle leg with bars). . . . . 23

2.9 The success of shape completion boosts shape structural understandings. We use Neural Shape Compiler to do *Point Cloud* $\Longrightarrow$ *Text* for both the partial chair and the complete chair. The ground-truth shape is on the top right. . . . . 25
2.10 *Partial Point Cloud* $\Longrightarrow$ *Complete Point Cloud*. Numbers are Chamfer distance $\downarrow$ with multiplying $10^2$. . . . . 25
2.11 *Point Cloud* $\Longrightarrow$ *Text* and *Point Cloud* $\Longrightarrow$ *Program* results over the input Hilbert cube point clouds by Shape Compiler. One description per sentence. . . . . 26
3.1 Cap3D provides detailed descriptions of 3D objects by leveraging pretrained models in captioning, alignment, and LLM to consolidate multi-view information. Two views of 3D objects are shown here, Cap3D uses eight. Additional examples are available in Appendix B.2. . . . . 29
3.2 **Overview of Cap3D**. Left to Right: (1) Render 3D objects from $M = 8$ camera angles to capture object details (2) Generate $N = 5$ image captions per rendered image using BLIP2; (3) Select one caption for each image based on its similarity to the image encoding using CLIP; (4) Use GPT4 to consolidate all selected captions into a final, summary of the object. . . . . 36
3.3 **Objaverse Caption Comparison.** Human captions and Internet metadata frequently contain limited detail. Cap3D captions typically have longer length and more detail. Vocabular size comparisons among Cap3D, BLIP2, and human captions are included in Appendix B.5. . . . . 38
3.4 **ABO Automated Geometric Description**. **Left:** Human descriptions provide more detailed geometry than automated captions. With careful prompting, *Cap3D (QA)* can match human-level detail. **Right:** The high peak of Metadata is cropped, which otherwise obscures other curves. . . . . 39
3.5 **Objaverse Caption Ablations**. GPT produces longer and more detailed captions than BLIP2; CLIP tends to prune incorrect details and reduces length slightly. . . 41
3.6 **Text-to-3D results.** Finetuning on Cap3D captions can significantly improve results. Additional examples are available in Appendix B.3. . . . . 45
3.7 An failed case. The caption under each rendered image are generated by BLIP2 + filtered by CLIP. The inaccurate content are highlighted with colors. GPT4 + CLIP cannot fix the error generated by BLIP2 and result in a fuzzy description. . 46
3.8 An failed case. The caption under each rendered image are generated by BLIP2 + filtered by CLIP. The inaccurate content are highlighted with colors. The various views contain inaccurate information. The associated details, roughly described, fail to accurately depict the indoor scene. . . . . 47
4.1 DiffuRank enhances caption accuracy and reduces hallucinations by prioritizing key rendered views (green box), contrasting the accidental views *Freeman* (1994) (red box) that cause errors. Surprisingly, using fewer views (6 vs. 28) not only saves computational resources but also may yield more accurate and detailed outcomes (the middle example) by countering the uncertainty caused by excessive views. . . . . 49

4.2 The left row features the top-6 views as ranked by DiffuRank, while the right row displays the bottom-6. Comparative analysis shows that the top-6 views generally uncover more characteristics of the object compared to the bottom-6. This finding underscores DiffuRank's capability to identify views that more accurately represent the features of the 3D object. More randomly sampled results are included in Appendix B.5. . . . . 50
4.3 Methods overview. Both Cap3D and our method render input 3D objects into multiple views for caption generation (green steps). However, while Cap3D consolidates these captions into a final description (blue steps), our method employs a pre-trained text-to-3D diffusion model to identify views that better match the input object's characteristics. These selected views are then processed by a Vision-Language Model (VLM) for captioning (orange steps). . . . . 54
4.4 We utilized both grey background + ray-tracing render engine (left images) and transparent background + real-time render engine (right images) for rendering, discovering that the effectiveness of each varies. We noticed DiffuRank can select the views with the appropriate rendering that highlight object features. . . . . 57
4.5 Mean and Max clip score distribution for Cap3D captions and their 8 rendering images. Selected thresholds are the two red dash lines via our annotated validation set. . . . . 59
4.6 Number of words in caption. . . . . 60
4.7 Number of n-grams for captions generated by different methods. . . . . 60
4.8 Each row represents a matched pair, and the accompanying text beneath it is the description. . . . . 65
5.1 **Sample data from ViLP.** For the same question, ViLP provides three distinct images and corresponding answers (upper-left corner). All questions follow a consistent structure, combining a distractor fact with a question. The *Prior Answer* (first column) can be directly inferred from the question, while *Test Answers* (second & third column) rely on visual cues. Our answers are designed to be single words, and both the model and human evaluators are tasked with open-domain answering, rather than selecting from predefined options. To support this, we have developed a robust synonym and plural detection pipeline, ensuring that open-ended responses do not hinder the evaluation process. This approach also enables evaluation without relying on LLMs. . . . . 69
5.3 **Qualitative examples.** We show the results from GPT-4o, Claude-3.5-Sonnet, Gemini-1.5-Pro, and Llama-3.2-Vision-90B for some challenging cases. Please refer to Appendix D.2.6 for categorized failure case analysis. . . . . 76
5.4 **Comparison of benchmark scores under different image transformations.** Solid line and dotted line refer to ViLP$^{F}$-Score and ViLP$^{F}$-Prior, respectively. . . 80
5.5 **Qualitative results before and after removing** distactor facts. GPT-4o and LLaVA-1.5-13B models yield completely opposite behaviors. . . . . 80
6.1 **Image-LoRA vs. Standard LoRA.** Comparison of accuracy and adapter-only training FLOPs on ScreenSpot-Pro. Points are connected across increasing input-text:image token ratios (1:2 $\rightarrow$ 1:5), where we control the ratios by dynamically adjusting image sizes. FLOPs are token-limited and computed as forward + backward multiply–adds. . . . . 87

6.2 **Overview. Left:** We evaluate Image-LoRA on grounded vision–language reasoning datasets, including ScreenSpot-Pro *Li et al.* (2025b) and RefCOCO *Kazemzadeh et al.* (2014), where the model takes a text query with its system prompt and outputs a point indicating the referred object. We further evaluate on the pure text reasoning dataset GSM8K *Cobbe et al.* (2021) to confirm that Image-LoRA does not affect pure text reasoning, and on ViLP *Luo et al.* (2025a) containing VQAs both aligned with and against language priors to show that Image-LoRA may mitigate shortcutting through such priors. **Right:** Image-LoRA is applied only to selected heads along the value path of attention layers within the visual-token span (the pink area). . . . . 88
6.3 **Top**: Share one $A$ per layer across its selected heads; learn $B^{(h)}$ only for the selected heads. **Bottom**: For the selected heads, we update the *value* vectors $v_t$ of the attention layer only on the visual-token span $\mathscr{I}_v$ and not on the text-token positions. . . . . 94
6.4 Head selection for Qwen2.5-VL-7B under a input-text:image token ratio of 1:2. . 100
7.1 **Overview**. When GUI agents encounter execution errors, we instruct the model to focus on a specific point of interest and extract multiple sub-regions around this focal point (1). The agent then independently generates candidate actions for each sub-region. Actions that interact with a specific coordinate are marked with pink-star landmarks (e.g., "click") to visually indicate the relevant location, along with action numbers (2). We retain the pink-star landmarks annotated with history indices to track RegionFocus's past interactions for diverse exploration (3). . . . 109
7.2 **Overview of integrating RegionFocus into GUI agent pipelines.** The standard inference process (blue arrow) takes input information and continuously predicts subsequent actions. When the GUI agent encounters errors, RegionFocus (green arrow) activates, proposing focal points to extract targeted sub-regions. Actions are then predicted individually for these sub-regions and aggregated into a single refined action for standard inference. . . . . 111
7.3 **Image-as-Map records temporal information.** We place numbered pink stars in the image as visual landmarks to indicate previously focused points. Each time a RegionFocus attempt fails (i.e., action takes no effect), we add a new pink star at the attempted location. Once an action successfully takes effect, we refresh the history and remove any existing landmarks. . . . . 116
7.4 **Qualitative results - Screenspot-Pro.** In this example, the agent successfully rejects the initial action via self-VLM-judge and proposes a correct grounding point based on the zoomed-in view. More qualitative results are listed in Appendix B. . . . . 119
7.5 **RegionFocus.** We illustrate how RegionFocus reduces background noise by emphasizing salient regions of an image. The mouse pointer indicates the agent's initial action prediction, which is suboptimal. The second image is zoomed in, reducing distracting details and allowing the agent to focus on the relevant information. . . . . 120

7.6 **Image-as-Map.** We show that action aggregation, enhanced by the proposed image-as-map, effectively distinguishes subtle coordinate differences between elements. Star landmarks are generated during RegionFocus prior to aggregation; the two landmarks at the top left correspond to the home and search buttons. . . . 120
7.7 **Ablation studies.** (1) RegionFocus can natively use the SAM to generate bounding boxes. (2) how image-as-map helps highlight subtle differences of different GUI elements. . . . . 122
7.8 **Ablation study – inference steps**: higher limits yield further improvements, though the benefits gradually decay. . . . . 124
A.1 Random examples in adjusted Text2Shape dataset. One description per sentence. 132
A.2 Random examples in adjusted ShapeGlot dataset. One description per sentence. . 133
A.3 Random examples in adjusted ABO dataset. One description per sentence. . . . . 134
A.4 The right three shapes are reconstruction results of random codes and the left two shapes are reconstruction results of indexed codes illustrated in the text. . . . . 140
A.5 A random example of interpolating latent codes between two resource shapes of our proposed *Point*VQVAE. . . . . 141
A.6 Random reconstruction examples of *Point*VQVAE. . . . . 142
A.7 *Point Cloud* $\Longrightarrow$ *Program* performance versus the shape generated sample numbers. . . . . 143
A.8 Generated shapes by Neural Shape Compiler given different text prompts. Shape Compiler can generate multiple shapes given a text query. . . . . 144
A.9 Generated shapes by Neural Shape Compiler given different text prompts. Shape Compiler can generate multiple shapes given a text query. . . . . 144
A.10 *Point Cloud* $\Longrightarrow$ *Text* results by Neural Shape Compiler given different point clouds. Shape Compiler can generate multiple captions given a point cloud query. One description per sentence. . . . . 145
A.11 Directly comparing the performance of Neural Shape Compiler with CLIP-Forge (*Sanghi et al.*, 2022) results on a simple text prompt. . . . . 145
A.12 Directly comparing the performance of Neural Shape Compiler with CLIP-Forge (*Sanghi et al.*, 2022) results on a simple text prompt. . . . . 146
A.13 *Point Cloud* $\Longrightarrow$ *Text* and *Point Cloud* $\Longrightarrow$ *Program* results by Neural Shape Compiler given the input point clouds. One description per sentence. . . . . 146
B.1 Random 3D captioning examples generated by Cap3D. Two views of 3D objects (Objaverse *Deitke et al.* (2023a)) are shown here, Cap3D uses eight. . . . . 148
B.2 Random 3D captioning examples generated by Cap3D. Two views of 3D objects (Objaverse *Deitke et al.* (2023a)) are shown here, Cap3D uses eight. . . . . 149
B.3 Random 3D captioning examples generated by Cap3D. Two views of 3D objects (Objaverse *Deitke et al.* (2023a)) are shown here, Cap3D uses eight. . . . . 150
B.4 Random 3D captioning examples generated by Cap3D. Two views of 3D objects (Objaverse *Deitke et al.* (2023a)) are shown here, Cap3D uses eight. . . . . 150
B.5 Random 3D captioning examples generated by Cap3D. Two views of 3D objects (Objaverse *Deitke et al.* (2023a)) are shown here, Cap3D uses eight. . . . . 151
B.6 Random 3D captioning examples generated by Cap3D. Two views of 3D objects (Objaverse *Deitke et al.* (2023a)) are shown here, Cap3D uses eight. . . . . 151

B.7 Comparative Analysis: Cap3D Generated Caption vs Human-Annotated Caption vs Objaverse Metadata *Deitke et al.* (2023a). Two views of 3D objects are shown here, Cap3D and human use eight. . . . 152
B.8 Comparative Analysis: Cap3D Generated Caption vs Human-Annotated Caption vs Objaverse Metadata *Deitke et al.* (2023a). Two views of 3D objects are shown here, Cap3D and human use eight. . . . 152
B.9 Text-to-3D results. The top text prompt and "Reference" are from our test set. We fine-tune the left 5-column methods on Cap3D-generated captions. The detailed setting and methods are described in §3.5.3. . . . 153
B.10 Text-to-3D results. The top text prompt and "Reference" are from our test set. We fine-tune the left 5-column methods on Cap3D-generated captions. The detailed setting and methods are described in §3.5.3. . . . 153
B.11 Text-to-3D results. The top text prompt and "Reference" are from our test set. We fine-tune the left 5-column methods on Cap3D-generated captions. The detailed setting and methods are described in §3.5.3. . . . 154
B.12 Text-to-3D results. The top text prompt and "Reference" are from our test set. We fine-tune the left 5-column methods on Cap3D-generated captions. The detailed setting and methods are described in §3.5.3. . . . 155
B.13 Text-to-3D results. The top text prompt and "Reference" are from our test set. We fine-tune the left 5-column methods on Cap3D-generated captions. The detailed setting and methods are described in §3.5.3. . . . 155
B.14 **ABO Caption Instructions.** . . . 160
B.15 **Objaverse Caption Instructions.** . . . 161
B.16 **A/B Instructions: Objaverse Captions.** . . . 163
B.17 **A/B Instructions: ABO Captions.** . . . 164
C.1 Comparisons between our captions and Cap3D captions. . . . 169
C.2 Comparisons between our captions with Cap3D captions. . . . 169
C.3 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . 170
C.4 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . 170
C.5 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . 171
C.6 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . 171
C.7 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . 171
C.8 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . 172
C.9 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . 172
C.10 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . 172
C.11 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . 173

C.12 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 173
C.13 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 173
C.14 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 174
C.15 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 174
C.16 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 174
C.17 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 175
C.18 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 175
C.19 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 175
C.20 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 176
C.21 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 176
C.22 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 176
C.23 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 177
C.24 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 177
C.25 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 177
C.26 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 178
C.27 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 178
C.28 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 178
C.29 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 179
C.30 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 179
C.31 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 179
C.32 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 180
C.33 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 180

C.34 We compare captions through random sampling, including those generated by our method, by Cap3D, and those authored by humans. . . . . 180
C.35 We evaluate captions by randomly sampling and comparing them across different methods: our approach (Top 6-views), using the bottom 6-views, utilizing all 28-views, and employing horizontal 6-views. . . . . 181
C.36 We evaluate captions by randomly sampling and comparing them across different methods: our approach (Top 6-views), using the bottom 6-views, utilizing all 28-views, and employing horizontal 6-views. . . . . 181
C.37 We evaluate captions by randomly sampling and comparing them across different methods: our approach (Top 6-views), using the bottom 6-views, utilizing all 28-views, and employing horizontal 6-views. . . . . 182
C.38 We evaluate captions by randomly sampling and comparing them across different methods: our approach (Top 6-views), using the bottom 6-views, utilizing all 28-views, and employing horizontal 6-views. . . . . 182
C.39 We evaluate captions by randomly sampling and comparing them across different methods: our approach (Top 6-views), using the bottom 6-views, utilizing all 28-views, and employing horizontal 6-views. . . . . 183
C.40 We evaluate captions by randomly sampling and comparing them across different methods: our approach (Top 6-views), using the bottom 6-views, utilizing all 28-views, and employing horizontal 6-views. . . . . 183
C.41 We evaluate captions by randomly sampling and comparing them across different methods: our approach (Top 6-views), using the bottom 6-views, utilizing all 28-views, and employing horizontal 6-views. . . . . 183
C.42 We evaluate captions by randomly sampling and comparing them across different methods: our approach (Top 6-views), using the bottom 6-views, utilizing all 28-views, and employing horizontal 6-views. . . . . 184
C.43 We evaluate captions by randomly sampling and comparing them across different methods: our approach (Top 6-views), using the bottom 6-views, utilizing all 28-views, and employing horizontal 6-views. . . . . 184
C.44 We evaluate captions by randomly sampling and comparing them across different methods: our approach (Top 6-views), using the bottom 6-views, utilizing all 28-views, and employing horizontal 6-views. . . . . 184
C.45 We evaluate captions by randomly sampling and comparing them across different methods: our approach (Top 6-views), using the bottom 6-views, utilizing all 28-views, and employing horizontal 6-views. . . . . 185
C.46 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 185
C.47 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 186
C.48 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 186

C.49 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 187
C.50 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 187
C.51 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 188
C.52 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 188
C.53 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 189
C.54 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 189
C.55 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 190
C.56 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 190
C.57 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 191
C.58 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 191
C.59 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 192
C.60 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 192
C.61 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 193
C.62 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 193
C.63 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . . 194

C.64 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . 194
C.65 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . 195
C.66 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . 195
C.67 Randomly sampled DiffuRank comparisons. **Top-row**: Top 6-views selected by DiffuRank; **Middle-row**: Bottom 6-views selected by DiffuRank; **Bottom-row**: Horizontal 6-views. . . . 196
C.68 Failure cases: hallucination. . . . 197
C.69 Failure cases: BLIP2 captioning fails or alignment compute not accurate. . . . 198
C.70 Failure cases: for some scene renderings, our framework fails to capture meaningful characteristics. . . . 199
C.71 Example hive case. Caption are from ours and Cap3D. . . . 199
C.72 Randomly sampled Text-to-3D results. . . . 201
C.73 Randomly sampled Text-to-3D results. . . . 202
C.74 Randomly sampled Text-to-3D results. . . . 203
C.75 Randomly sampled Text-to-3D results. . . . 204
C.76 Randomly sampled Text-to-3D results. . . . 205
C.77 Randomly sampled Text-to-3D results. . . . 206
D.1 **Gradients difference between Image-DPO and Text-DPO.** For Text-DPO (*Rafailov et al.*, 2024), the model receives positive gradients (green arrows) for the preferred answer $A_w$ and negative gradients $A_l$ (red arrows) for the dispreferred answer. In contrast, our proposed Image-DPO approach applies positive gradients when the preferred image $I_w$ is input and negative gradients for the dispreferred image $I_l$, both based on the same output answer. . . . 210
D.2 **Randomly sampled data from ViLP.** . . . 214
D.3 **Randomly sampled data from ViLP.** . . . 215
D.4 **The prompt we used for generating text-prompt for image generation.** . . . 217
D.6 **Realistic image comparison.** Each image pair shows our original benchmark data on the left and a corresponding realistic example generated by GPT-4o on the right. . . . 219
D.5 **One prompt we used for potential QAs designs of ViLP** . . . 219
D.7 **Winoground Data Example.** Our benchmark is different from Winoground, as Winoground focuses on vision-linguistic compositional reasoning. Both captions and images are normal and satisfy common linguistic expectations and common sense. . . . 221
D.8 **Example of Whoops! Dataset.**Whoops! also has creative images. While unlike ours, its questions are common questions without strong language priors. . . . 222
D.9 **Example of HallusionBench**. HallusionBench also has questions which can be answered without images. While it is based on facts instead of stereotypes like ours. Moreover, its images are limited to *Chart*, *Map*, *OCR* and *Table*. . . . 223

D.10 **Failure cases analysis.** We list six typical failure cases generated by VLMs on ViLP. . . . . 227
D.11 **Overview of our data generation pipeline.** We begin with an image only, from which instructions are derived using a VLM. These instructions guide the creation of a new image or the modification of the existing one. The generated image is then processed by the VLM to generate QA pairs. Both the QA pair and the image are subsequently input back into the VLM to assess the quality of the answers. No human-written in-context examples are used throughout this process. . . . . 228
D.12 **Randomly sampled generation examples.** Our proposed data generation pipeline utilizes pretrained model to generate diverse new images from existing seed image datasets. . . . . 229
D.13 **The prompt for instruction generation.** We ask the VLM to generate instructions for using pre-trained image models. . . . . 230
D.14 **The prompt for single-image QAs.** We ask the VLM itself to generate single-image QAs based on the generated images by pre-trained models. . . . . 231
D.15 **The prompt for instruction generation of Grounded-SAM.** We ask the VLM to generate designated instructions to use Grounded-SAM. . . . . 232
D.16 **Randomly sampled single-image QA generated by our data generation pipeline**.234
D.17 **Randomly sampled single-image QA generated by our data generation pipeline**.234
D.18 **Randomly sampled single-image QA generated by our data generation pipeline**.235
D.19 **Randomly sampled single-image QA generated by our data generation pipeline**.235
D.20 **Randomly sampled single-image QA generated by our data generation pipeline**.236
D.21 **Randomly sampled single-image QA generated by our data generation pipeline**.236
D.22 **Randomly sampled single-image QA generated by our data generation pipeline**.237
D.23 **Randomly sampled single-image QA generated by our data generation pipeline**.237
D.24 **Randomly sampled single-image QA generated by our data generation pipeline**.238
D.25 **Randomly sampled single-image QA generated by our data generation pipeline**.238
D.26 **Randomly sampled single-image QA generated by our data generation pipeline**.239
D.27 **Randomly sampled single-image QA generated by our data generation pipeline**.239
D.28 **Randomly sampled single-image QA generated by our data generation pipeline**.240
D.29 **Randomly sampled single-image QA generated by our data generation pipeline**.240
D.30 **Randomly sampled single-image QA generated by our data generation pipeline**.241
E.1 **Head selection patterns with different $\tau$.** The head selection procedure uses default hyper-parameters $\rho = 2$. All results are obtained on ScreenSpot-Pro using the 1:2 input-text:image token ratio. Intuitively, $\tau = 0$ yields an *approximately uniform* allocation across layers, and $\tau = 1$ makes the budget *proportional to mass* $\Phi_L$, We use $\tau = 0.5$ in our main experiments. For Qwen2.5-VL-7B with 28 layers and $K_{\text{sel}} = 28$, the $\tau = 0$ setting reduces to select exactly one KV head per layer — the one with the highest importance score. . . . . 250
E.2 **Head selection patterns with different $\rho$.** The head selection procedure uses default hyper-parameters $\tau = 0.5$. All results are obtained on ScreenSpot-Pro using the 1:2 input-text:image token ratio. We use $\rho = 2$ in our main experiments. $\rho = 1$ reduces exactly to pure top-$k_L$ by importance, while larger $\rho$ allows more diversity at a small cost in $I(h)$. . . . . 251

E.3 **Head selection for Qwen2.5-VL-72B under a input-text:image token ratio of 1:2.** . . . 254
E.4 **Comparison between the correlation-map–based approach and our proposed head-selection method.** . . . 255
E.5 **Head selection for LLaVa-Next-7B under 1:2 text:image ratio.** . . . 255
E.6 **Head selection patterns under different head budgets.** The head selection procedure uses default hyper-parameters of $\tau = 0.5$ and $\rho = 2$. All results are obtained on ScreenSpot-Pro using the 1:2 input-text:image token ratio. . . . 255
E.7 **Head selection for Qwen2.5-VL-7B across different input-text:image token ratios on ScreenSpot-Pro.** Although ratios (and thus image resolutions) differ, the resulting head selections remain similar, with minor variations. . . . 256
F.1 **Histogram of Step Differences on WebVoyager**: Comparing UI-TARS-72B with and without RegionFocus across 400+ trajectories. . . . 258
F.2 **Qualitative Results - RegionFocus.** In these two examples, we illustrate how RegionFocus reduces background noise by emphasizing salient regions of an image. The mouse pointer indicates the agent's initial action prediction, which is suboptimal in both cases. **Left pair of images**: The green window in the second image marks the zoomed-in region. By focusing on this region, we naturally cut out the distracting portion of the first image. **Right pair of images**: The second image is zoomed in, significantly reducing distracting details. This allows the agent to focus on the region with more relevant information. . . . 259
F.3 **Qualitative results - image-as-map.** These examples demonstrate how action aggregation, enhanced by the proposed image-as-map, helps distinguish subtle coordinate differences between target elements. The mouse pointer indicates the agent's initial predictions, which were incorrect in both cases. Each star-like landmark is generated during the RegionFocus process before action aggregation. **Left pair of images**: The two top-left landmarks correspond to the home and search buttons. **Right pair of images**: the landmarks correspond to different options in a dropdown menu. . . . 259
F.4 **Qualitative results - Screenspot-Pro.** In one example from our evaluation, the system successfully rejects the initial action and proposes a correct grounding point based on the zoomed-in view. . . . 262
F.5 **Qualitative results - Screenspot-Pro.** In one example from our evaluation, the system successfully rejects the initial action and proposes a correct grounding point based on the zoomed-in view. . . . 262
F.6 **Qualitative results - Screenspot-Pro.** In one example from our evaluation, the system successfully rejects the initial action and proposes a correct grounding point based on the zoomed-in view. . . . 263

# LIST OF TABLES

**Table**


2.1 *Text* $\Longrightarrow$ *Point Cloud*. MMD (quality), AMD (fidelity), and TMD (diversity) are multiplied by $10^3$, $10^3$ and $10^2$, respectively. . . . . 19

2.2 The rightmost column is the cosine similarity between the CLIP text embeddings (*Radford et al.*, 2021b) of the left two texts. Cosine Similarity $= \frac{f_1 \cdot f_2}{\max(\|f_1\|_2 \cdot \|f_2\|_2, \varepsilon)}$, where $f_1$ and $f_2$ are the CLIP embedding of Text1 and Text2, respectively. $\varepsilon$ is 1e-6 to avoid division by zero. . . . . 20

2.3 *Point Cloud* $\Longrightarrow$ *Text*. BLEU, CIDER, and ROUGE measure quality. Dist-1 and Dist-2 measure diversity. . . . . 22

2.4 *Point Cloud* $\Longrightarrow$ *Program*. CD is multiplied by $10^2$. Avg denotes average number among categories. . . . . 24

3.1 Cap3D is ***better, cheaper, and faster*** than crowdsourced annotation. Use 36k responses across 22k objects for A/B testing; 8A40s on a cloud platform for speed and cost computations. . . . . 32

3.2 **Ethical Filtering Analysis.** We manually detect faces and NSFW content to validate automated filtering. 16 of 17 missed face detections were sports cards. . 37

3.3 **Objaverse Captions Evaluations**. *Cap3D* outperforms *human* and *Metadata*; *BLIP2*, *GPT4*, and *CLIP* are all important to performance. We report 95% confidence interval and use 5k objects. . . . . 41

3.4 **ABO Fine-Grained Geometry Captions**. Cap3D (QA) performs best; crowdsourced beats captioning alone. . . . . 43

3.5 **Text-to-3D: Human Captions**. Cap3D captions are better than human on the 30k set. Finetuning on Cap3D full set performs best. . . . . 43

3.6 **Text-to-3D on Objaverse**. Finetuning improves FID over pretrained performance across models. CLIP metrics of *Stable Diffusion* increase; CLIP metrics of *Point·E* increase significantly. . . . . 44

3.7 **Text-to-3D: Optimization Baselines**. Overfitting via CLIP leads to higher CLIP-based scores than ground truth; ViLT score is more fair. . . . . 44

4.1 **Objaverse Captions Evaluations**. All A/B testing represents captions from other methods vs. ours. We tested on 5k objects. . . . . 62

4.2 **Text-to-3D Finetuning experiments**. . . . . 63

4.3 Accuracy comparison among various VLMs, CLIP, and our method. . . . . 65

5.1 **Category Statics**. . . . . 74

5.2 **Benchmarking on ViLP**. Please refer to the left text for symbol definitions. † indicates the model often fails to follow the instructions. . . . . 79
5.3 **Effectiveness of Image-DPO on General VQA benchmarks.** . . . . . 81
5.4 **Benchmark Performance with More Fine-Grained Categories.** We extended our evaluation system to include three categories—*Correct*, *Wrong*, and *Invalid*. "Invalid" responses (not single-word) are further split into *Invalid-Correct* (noted as Invalid-✓) and *Invalid-Wrong* (noted as Invalid-×) based on GPT assessment. 81
5.5 **Comparisons of Image-DPO on ViLP.** . . . . . 84
6.1 **Main results across input text:image ratios.** FLOPs (G) denotes the total adapter-only training-time floating-point operations, measured in billions ($10^{-9}$). The average fine-tuned parameter count for Image-LoRA for 7B and 72B models are, respectively, 629,781 and 3,011,629. The respective parameter counts for Std-LoRA are 2,523,192 and 16,384,160, respectively. . . . . 98
6.2 **Head selection methods ablation on Qwen2.5-VL-7B.** All methods adapt $K_{\text{sel}}$=28 heads. Entries are accuracy (%). . . . . 103
6.3 **KV-budget ablation on Qwen2.5-VL-7B.** We vary $K_{\text{sel}} \in \{1, 7, 28, 112\}$. Entries are accuracy (%). . . . . 103
6.4 **Updating $K$ in ImageLoRA + different LoRA keys for Std-LoRA.** Entries are accuracy (%). Image-LoRA with $V$ only is our proposed method. . . . . 104
6.5 **Effect of selection-size normalization on Qwen2.5-VL-7B.** All runs use the proposed head selection procedure, $K_{\text{sel}}$=28, $V$-only, $r$=8. Entries are accuracy (%). 104
7.1 **Comparison of various models on ScreenSpot-Pro.** *Li et al.* (2025b). Our proposed RegionFocus helps the UI-TARS-72B *Qin et al.* (2025) model achieve a 31.8% improvement, while Qwen2.5-VL-72B *Bai et al.* (2025) sees a 28.9% gain, thereby achieving state-of-the-art performance. Additionally, integrating RegionFocus into UI-TARS-7B allows it to surpass the performance of the substantially larger UI-TARS-72B model. . . . . 116
7.2 **Comparison of various models on WebVoyager** *He et al.* (2024). For each automatic evaluation, we run GPT evaluator three times to calculate the performance mean and standard deviation. We evaluated our method on UI-TARS *Qin et al.* (2025) and Qwen2.5-VL *Bai et al.* (2025), consistently observing performance improvements. The performance on Google Search was significantly affected by bot blocking - See text for details. . . . . 118
7.3 **Ablation study of RegionFocus components.** Results are reported on a subset of WebVoyager, with higher scores indicating better performance. . . . . 121
A.1 Point Cloud auto-encoding performance on ShapeNet. CD is multiplied by $10^3$. X in *Point*VQVAE final-X means how many points we sampled in the last layer of the encoder. Globalpool indicates only having one point with the global receptive field. All the models besides *Point*VQVAE are trained on ShapeNet (*Chang et al.*, 2015) shapes. *Point*VQVAE is trained on all our 3D shapes. . . . . 141
B.1 **ABO Automated Caption Evaluations**. Automated captions are a poor measure of performance on ABO as (1) many objects are similar, making retrieval difficult; (2) automated captioning does not describe geometry well, so we should not expect automated image-caption alignment to describe geometrically correct captions well. 156
B.2 N-gram Comparison among Cap3D, BLIP2, and Human Captions . . . . . 157

D.1 **Impacts of Realistic Styles.** Each value represents the change in correctness when replacing the original images with realistic ones (*Realistic* - *Original*). Negative values indicate a drop in performance, suggesting increased task difficulty. The metric definitions are provided in the beginning of Section 5.5. . . . . 220
E.1 **Example image resolutions for each input text:image token ratio.** The average image token counts (**#Image Tokens**) correspond to Table 6.1. We provide two example image resolutions for reference. . . . . 248
E.2 **Ablations on layerwise budgeting $\tau$ and diversity factor $\rho$ on Qwen2.5-VL-7B.** Columns report accuracy (%) at different text:image token ratios. Defaults underlined ($\underline{\tau = 0.5, \rho = 2.0}$). . . . . 249
E.3 **LoRA rank ablation on Qwen2.5-VL-7B.** All methods adapt $K_{\text{sel}}$=28 heads. Entries are accuracy (%). . . . . 250
E.4 **Trainable parameters (M) and per-example FLOPs (G) - Qwen2.5-VL-7B**. We test on ScreenSpot-Pro across input-text:image token ratios for the Qwen2.5-VL-7B model. Image-LoRA is V-only configurations. . . . . 253
E.5 **Trainable parameters (M) and per-example FLOPs (G) - Qwen2.5-VL-72B**. We test on ScreenSpot-Pro across input-text:image token ratios for the Qwen2.5-VL-72B model. Image-LoRA is V-only configurations. . . . . 253

# LIST OF APPENDICES

**Appendix**

A. Supplementary Material for Neural Shape Compiler . . . . . 131

B. Supplementary Material for Scalable 3D Captioning with Pretrained Models . . . . 147

C. Supplementary Material for View Selection for 3D Captioning via Diffusion Ranking 167

D. Supplementary Material for Probing Language Priors on Visual Reasoning in VLMs 209

E. Supplementary Material for Towards Minimal Fine-Tuning of VLMs . . . . . 242

F. Supplementary Material for Visual Test-time Scaling for GUI Agent Grounding . . 257

# ABSTRACT

Visual intelligence—the ability to convert visual observations into meaningful representations, predictions, and actions—is fundamentally grounded in structure. Images arise from compositional 3D scenes projected through cameras; objects and parts combine according to physical and semantic regularities; multi-view observations are linked by geometry; and tasks often require locating a small, query-relevant subset of evidence within cluttered inputs. Similarly, modern vision–language models (VLMs) are themselves structured systems: transformers route computation through token spans, attention heads, and specialized pathways that can be exploited or constrained. Despite this pervasive structure, common training and modeling pipelines often treat both data and model components as homogeneous, leaving efficiency on the table and amplifying brittle correlations such as language priors or background clutter.

This thesis argues that *explicitly leveraging structure*—in representations, data construction, objectives, and model internals—yields visual intelligence that is both more scalable and more grounded. Structure in data can guide supervision; structure in objectives can diagnose and correct model biases; and structure in model architecture can enable selective, efficient adaptation that targets the components most relevant to a given behavior. Although the principle applies broadly across visual computing (e.g., scene understanding, video reasoning, embodied perception), this dissertation develops it through three representative domains: *3D understanding*, where multi-view geometry and compositional shape provide natural structural axes; *vision–language reasoning*, where structured diagnostics and objectives can expose and suppress shortcutting, and where internal model structure (e.g., attention-head specialization and token-span roles) enables selective, efficient adaptation; and *GUI interaction*, where task success hinges on localizing relevant interface elements within cluttered screens.

Specifically, I (1) represent 3D structure through a unified discrete code that enables transformation between text, point clouds, and shape programs; (2) scale 3D supervision via multi-view captioning and structure-aware view selection; (3) diagnose visual–language shortcutting with out-of-distribution, prior-conflicting visual questions and introduce a preference-optimization (DPO) training objective that suppresses these shortcuts; (4) develop a minimal VLM fine-tuning recipe that applies low-rank updates only to the value pathway within the visual-token span, and only for a selected subset of attention heads—reducing adaptation cost while preserving text-only reasoning performance; and (5) propose a test-time scaling strategy that improves GUI agents by reducing background clutter. Together, these results demonstrate that structural approaches can serve as both supervision and constraint for scalable, grounded visual intelligence.

# CHAPTER 1

# Introduction

Visual intelligence—the capacity to transform visual observations into meaningful representations, predictions, and actions—pervades nearly every area of modern AI: from perceiving 3D scenes (*Schönberger et al.*, 2016) and understanding images through language (*Johnson*, 2018), to navigating graphical interfaces (*Yao et al.*, 2022), driving vehicles (*Bojarski et al.*, 2016), and controlling robots (*Brohan et al.*, 2022). Recent progress in vision–language modeling has largely been driven by scale—large image–text corpora, transformer architectures, and foundation-model training pipelines (*Vaswani et al.*, 2017; *Dosovitskiy et al.*, 2021; *Radford et al.*, 2021b; *Alayrac et al.*, 2022b; *Li et al.*, 2023d; *OpenAI*, 2023b; *Liu et al.*, 2023d; *Yang et al.*, 2025a). Yet the problems underlying visual intelligence are not solved by scale alone; they are fundamentally *structured*—as evidenced by the fact that even the best commercial VLMs still fall short of specialized vision models on specific downstream visual tasks (*Carion et al.*, 2025; *Bochkovskii et al.*, 2024; *Harley et al.*, 2025; *Teed and Deng*, 2020).

Structure manifests at multiple levels. At the *data level*, images are not arbitrary pixel arrays: they arise from compositional 3D objects and scenes projected through cameras, and multiple viewpoints of the same scene are coupled by geometry (*Hartley and Zisserman*, 2003; *Mildenhall et al.*, 2021). Videos add temporal structure—object persistence, motion continuity, and causal dynamics (*Rockwell et al.*, 2025; *Finn et al.*, 2016). At the *task level*, success often depends on a small, query-relevant subset of visual evidence (e.g., the pixels that justify a grounding decision, or

the interface element relevant to an action), rather than the entire input (*Xu et al.*, 2015; *Selvaraju et al.*, 2017). At the *model level*, modern vision–language models (VLMs) are not monolithic: transformer computation is routed through distinct token spans and attention heads, and different components can play different functional roles (*Vaswani et al.*, 2017; *Michel et al.*, 2019; *Voita et al.*, 2019; *Kang et al.*, 2025). Structure is everywhere—in the physical world that generates images, in the tasks we wish to solve, and in the architectures we use to solve them.

Despite this pervasive structure, common training and adaptation pipelines often treat both data and model components as largely homogeneous *Alayrac et al.* (2022b); *Liu et al.* (2023d); *Yang et al.* (2025a). This mismatch has practical consequences. On the data side, it can waste supervision by ignoring geometric relationships across views and by collapsing rich 3D regularities into single-view learning. On the modeling side, uniform fine-tuning can be unnecessarily expensive and can entangle vision-specific adaptation with text-only reasoning. More subtly, homogeneous training can amplify brittle correlations: for example, vision–language models may over-rely on language priors that correlate with answers rather than grounding decisions in visual evidence (*Luo et al.*, 2025a; *Guan et al.*, 2023). In interactive settings, systems can likewise latch onto background clutter or spurious UI cues instead of the small set of relevant regions.

This dissertation argues that *explicitly leveraging structure*—in representations, data construction, training objectives, and parameter updates—yields visual intelligence that is both more scalable and more grounded. The central theme is not to add structure for its own sake, but to expose and exploit the structure that is already present: in the 3D world that generates images, in the task-dependent evidence needed for decisions, and in the internal modularity of multimodal transformers. While this principle applies broadly—to scene understanding, video reasoning, embodied perception, medical imaging, and beyond—this dissertation develops and validates it through three representative domains: (i) *3D understanding*, where multi-view geometry and compositional shape provide natural structural axes; (ii) *vision–language reasoning*, where structured diagnostics and objectives can expose and suppress shortcutting, and where internal model structure enables selective, efficient adaptation; and (iii) *GUI interaction*, where task success hinges on localizing relevant interface

elements within cluttered screens. The underlying ideas—exploiting data geometry, diagnosing and correcting model biases, and selectively adapting model components—extend naturally to other areas of visual intelligence.

## 1.1 Thesis Overview

**Structure as Representation and Supervision for 3D (Chapters 2–4).** A first axis of structure is the 3D nature of the visual world. Modern 3D learning uses many representations (e.g., point clouds and programmatic descriptions) that each highlight different aspects of shape (*Qi et al.*, 2017a; *Tian et al.*, 2019). In Chapter 2, I introduce *Neural Shape Compiler* (*Luo et al.*, 2023a), published at TMLR 2023, which makes 3D structure explicit via a unified discrete code that supports translation between text, point clouds, and program-like shape descriptions, enabling a single model to act as a "compiler" across modalities. In Chapter 3, I study how to scale 3D supervision by constructing large-scale multi-view captions from existing 3D assets, yielding the Cap3D pipeline and dataset (*Luo et al.*, 2023b), published at NeurIPS 2023. This direction is enabled by the growing availability of large 3D repositories (*Deitke et al.*, 2023b) and advances in generative modeling (*Ho et al.*, 2020; *Rombach et al.*, 2022; *Jun and Nichol*, 2023). In Chapter 4, I further exploit multi-view structure by proposing structure-aware view selection (DiffuRank) (*Luo et al.*, 2024), published at ECCV 2024, using generative-model signals to prioritize informative views for supervision.

**Structure as a Diagnostic and Training Signal for VLM Grounding (Chapter 5).** A second axis of structure concerns *how* vision and language interact during reasoning. Because natural data contains strong answer biases, models can succeed by following language priors rather than grounding in the image (*Agrawal et al.*, 2016; *Goyal et al.*, 2017). In Chapter 5, I introduce a diagnostic framework (ViLP) for probing such visual–language priors with controlled, prior-conflicting questions (*Luo et al.*, 2025a), published at ICML 2025. Building on this diagnosis, I also study how to use *structured objectives* to suppress shortcutting, including preference-based optimization inspired by recent advances in direct preference optimization (*Rafailov et al.*, 2024).

**Structure Inside the Model: Selective Adaptation Rather Than Uniform Fine-Tuning (Chapter 6).** A third axis of structure is internal to multimodal transformers. Attention heads are known to exhibit redundancy and specialization, and not all heads contribute equally to a given behavior (*Michel et al.*, 2019; *Voita et al.*, 2019). Likewise, multimodal prompts contain distinct token regions (system/prompt text, visual tokens, and output text) whose roles differ during training. In Chapter 6, I develop *Image-LoRA*, a minimal fine-tuning recipe that applies low-rank updates (*Hu et al.*, 2021) selectively: only along the value pathway, only within the visual-token span, and only for a chosen subset of attention heads. This structured adaptation reduces training cost while preserving text-only reasoning behavior, aligning the fine-tuning mechanism with the model's internal organization.

**Structure at Test Time for Interactive Visual Agents (Chapter 7).** Finally, interactive GUI and web tasks introduce a concrete form of structure: success often depends on identifying and acting on a small set of relevant interface elements within cluttered screens. Recent benchmarks highlight both the importance and difficulty of this setting (*Li et al.*, 2025a; *He et al.*, 2024; *Zhou et al.*, 2024a; *Koh et al.*, 2024). In Chapter 7, I introduce *RegionFocus* (*Luo et al.*, 2025b), published at ICCV 2025, a test-time scaling strategy that reduces background clutter by dynamically zooming in on task-relevant regions. This approach can optionally leverage general-purpose segmentation to further isolate actionable areas (*Kirillov et al.*, 2023), translating structural assumptions about "where the evidence is" into improved action grounding.

## 1.2 First published appearances of the described contributions

This dissertation is organized as a collection of research chapters, each based on a published or ongoing research project. The following papers form the basis of the chapters in this thesis:

- Chapter 2: T. Luo, H. Lee, and J. Johnson. Neural Shape Compiler: A Unified Framework for Transforming between Text, Point Cloud, and Program. In *Transactions on Machine Learning Research (TMLR)*, 2023.

- Chapter 3: T. Luo, C. Rockwell, H. Lee, and J. Johnson. Scalable 3D Captioning with Pretrained Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

- Chapter 4: T. Luo, J. Johnson, and H. Lee. View Selection for 3D Captioning via Diffusion Ranking. In *European Conference on Computer Vision (ECCV)*, 2024.

- Chapter 5: T. Luo, A. Cao, G. Lee, J. Johnson, and H. Lee. Probing Visual Language Priors in VLMs. In *International Conference on Machine Learning (ICML)*, 2025.

- Chapter 6: T. Luo, H. Lee, and J. Johnson. Image-LoRA: Selective Visual Fine-Tuning for Vision-Language Models. *arXiv*, 2025.

- Chapter 7: T. Luo, L. Logeswaran, J. Johnson, and H. Lee. Visual Test-time Scaling for GUI Agent Grounding. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

# CHAPTER 2

# Neural Shape Compiler: A Unified Framework for Transforming between Text, Point Cloud, and Program

*As outlined in the Introduction, structure is a unifying lens for building visual intelligence that is both scalable and grounded. This chapter focuses on the first axis:* ***structure as representation****. The 3D world admits many complementary descriptions—point clouds capture geometry, programs encode part-level regularities, and natural language conveys semantics. Rather than treating these as isolated modalities, we propose a unified framework that transforms freely among them, making explicit the compositional structure inherent in 3D shapes.*

## 2.1 Introduction

Humans understand 3D shapes from different perspectives: we perceive geometries, understand their composing parts and regularities, and describe them with natural language. Similarly, vision researchers designed different abstractions for 3D shapes, including (1) low-level geometric structures that plot detailed geometry such as point clouds (*Gruen and Akca*, 2005; *Qi et al.*, 2017a); (2) structure-aware representations that can tell shape parts and their relations, like shape programs (*Tian et al.*, 2019); (3) natural language to describe compositionality and functionality (*Çağdaş*,

This chapter is based on work published at TMLR 2023 (*Luo et al.*, 2023a).

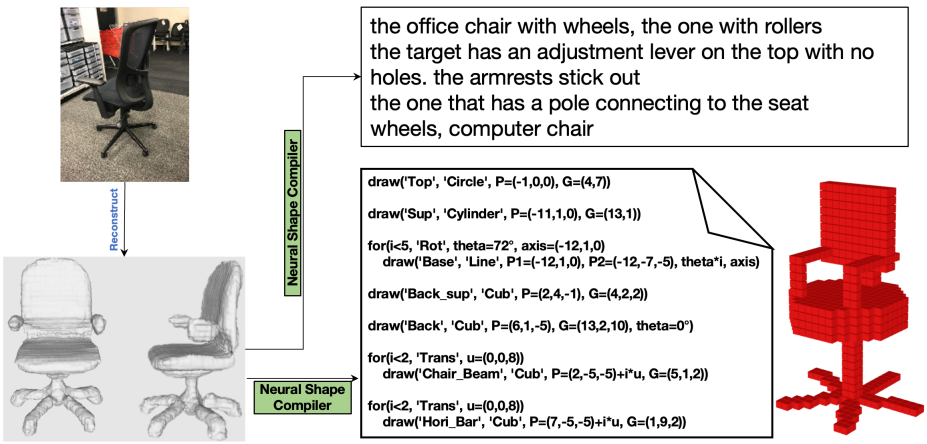



Figure 2.1: Example: Neural Shape Compiler helps obtain hierarchical information for the reconstruction result of a single image (*Zhang et al.*, 2018), including its structural descriptions, regularities, and how to assemble it.

1996; *Chang et al.*, 2014). Different shape abstractions convey complementary information and encode different characteristics allowing researchers to design specialized models for various tasks (*Mitra and Nguyen*, 2003; *Chaudhuri et al.*, 2020). This paper studies how to translate between *Text* $\Longleftrightarrow$ *Point Cloud* $\Longleftrightarrow$ *Program*. Such transformation mechanism can provide multi-level information about shape (Figure 2.1) to support various downstream tasks. Additionally, we demonstrate that combining the learning of different transformation branches within a unified framework leads to stronger representation learning and improved performance on each individual task.

Modern program compilers (*Lattner and Adve*, 2004; *Zakai*, 2011) turn source codes into intermediate representations (*IRs*), transform *IRs*, and decode them into other types of high-level programming languages. We take inspiration here and propose a unified architecture, ***Neural Shape Compiler***, to perform compilation between the three shape abstractions with neural networks. It converts all shape abstractions into unified discrete shape codes (i.e., IR) via their respective encoders. Then, *ShapeCode Transformer*, implemented as the standard Transformer (*Vaswani et al.*, 2017) for generality, transforms shape codes of one type to another in an autoregressive and probabilistic manner (*Ramesh et al.*, 2021). The probabilistic way helps us overcome the issue of ambiguity that different shape geometries can correspond to very similar program or text. Finally, the compiled shape code is decoded to the target shape abstraction through the corresponding

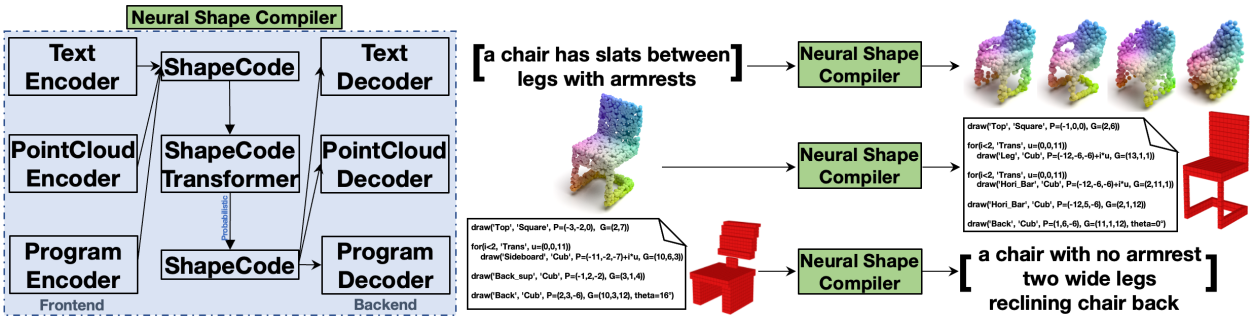



Figure 2.2: Overview, and three compilations conducted by Neural Shape Compiler, including *Text* $\Rightarrow$ *Point Cloud*, *Point Cloud* $\Rightarrow$ *Program*, and *Program* $\Rightarrow$ *Text*Footnote 1. Red shapes are the rendered results by executing the corresponding program. ShapeCode Transformer is shared across different compilations. Neural Shape Compiler is not limited to performing the compilations shown above, and benefits from joint training with all heterogeneous data and tasks.

decoder[1]. Figure 2.2 shows the entire process and three example compilations.

To turn point clouds with continuous coordinates into discrete shape code, we propose ***Point*VQVAE** inspired by (*Van Den Oord et al.*, 2017). Unlike traditional point cloud process models (*Qi et al.*, 2017a), our encoder has restricted receptive fields (*Coates and Ng*, 2011; *Luo et al.*, 2019) for learning shape part embeddings. Those part embeddings let the codebook (*Van Den Oord et al.*, 2017) encode the parts of the input 3D shape rather than the entire shape. The decoder then combines all the 3D part codes to reconstruct whole point clouds in a permutation-equivalent manner.

This paper focuses on modeling 3D shape structures. Our text data contains detailed structural descriptions and is devoid of color and texture, where we adjusted and annotated data from Text2Shape (*Chen et al.*, 2018), ShapeGlot (*Achlioptas et al.*, 2019a), and ABO (*Collins et al.*, 2021) datasets, resulting in 107,371 (*Point Cloud*, *Structure-Related Text*) pairs. We synthesized 120,000 (*Point Cloud*, *Program*) pairs (*Tian et al.*, 2019) for understanding shape parts and regularities with programs. Our experiments show that Neural Shape Compiler has the ability to generate point clouds from input text with geometric details, generate text descriptions of the structure of input point clouds, and generate programs for composing point clouds. Additionally, our experiments suggest the image-text pretraining model, CLIP (*Radford et al.*, 2021b), used by recent *Text* $\Longrightarrow$ *3D* methods (*Jain et al.*, 2021; *Sanghi et al.*, 2022; *Nichol et al.*, 2022) is *NOT* suitable for generating

[1] *Text* $\Longleftrightarrow$ *Program* is achieved in two-step generation *Text* $\Longleftrightarrow$ *Point Cloud* $\Longleftrightarrow$ *Program*. *Program* $\Rightarrow$ *Point Cloud* is deterministic via executing the programs

3D shapes with structural details.

Furthermore, we train a model using all the data from tasks *Text* $\Longrightarrow$ *Point Cloud*, *Point Cloud* $\Longrightarrow$ *Text*, and *Point Cloud* $\Longrightarrow$ *Program*, and train a separate limited version of the model for each task. Our experiments show that the jointly trained model consistently outperforms each limited version model on the corresponding tasks and beats existing baselines on all aforementioned tasks.

Neural Shape Compiler is extensible. We show a case where our framework is extended to perform Point Cloud completion by transforming partial Point Cloud code into complete Point Cloud code. Limitations and future work are discussed in Section 2.5. Our code, dataset, and pre-trained models will be released to facilitate future research in 3D multimodal learning.

## 2.2 Related Work

**Compiler:** Our proposed framework is inspired by the concept of computer program compiler (*Calingaert*, 1979; *Appel*, 2004; *Aho et al.*, 2007; *Aho and Ullman*, 2022). Classical compilers translate high-level program language (e.g., Pascal, C) into executable machine code with the help of assemblers and linkers (*Wirth et al.*, 1996; *Appel*, 2004). Compared to classical compilers, our framework is more similar to modern program compilers (e.g., LLVM (*Lattner and Adve*, 2004) and Emscripten (*Zakai*, 2011)) where the target code is not limited to machine code and can be high-level programming languages. Our framework (Figure 2.2) shares similar architectures with LLVM: LLVM uses front-ends (encoders) to turn the corresponding source codes into intermediate representations (ShapeCode) and decode them via back-ends (decoders). A major difference between our framework and modern program compilers is that our IR transformation process is probabilistic, whereas the process in a program compiler is a deterministic process with potential performance optimizations.

**Multimodal Learning:** With web-scale image-text data, our community achieved remarkable progress in multi-modal learning of 2D-text (*Li et al.*, 2019a; *Ramesh et al.*, 2021; *Radford et al.*, 2021b; *Patashnik et al.*, 2021; *Alayrac et al.*, 2022b; *Ramesh et al.*, 2022). However, due to the lack of large-scale 3D-text pairs and baseline systems (*He et al.*, 2017), there is slow progress in

3D-text modeling. Some recent works tried to leverage the progress in 2D-text multimodal learning (e.g., CLIP, Imagen (*Saharia et al.*, 2022) (*Radford et al.*, 2021b)) for 3D-text modeling (*Jain et al.*, 2021; *Zhang et al.*, 2021b; *Sanghi et al.*, 2022; *Liu et al.*, 2022a; *Poole et al.*, 2022; *Lin et al.*, 2022; *Nichol et al.*, 2022). However, our experiments suggest CLIP is *NOT* suitable for text-guided 3D generations once text contains structural details. Compared to them, this work studies the connections between 3D and text directly (*Chen et al.*, 2018; *Fu et al.*, 2022), as there are significant compositional connections between shape parts and words. Our work is closer to Text-to-Voxel works (*Chen et al.*, 2018; *Liu et al.*, 2022b), while none of them can generate desirable shapes corresponding to text prompt with levels of geometric details. *Mittal et al.* (2022) concurrently studied language-guided shape generation with learned autoregressive shape prior. Besides, the proposed framework is related to many-to-many universal deep learning frameworks (*Jaegle et al.*, 2021; *Reed et al.*, 2022).

**Generative Model:** *Point*VQVAE is inspired from VQVAE (*Van Den Oord et al.*, 2017) studying variational auto-encoder (*Kingma and Welling*, 2013) with discrete latent variables (*Salakhutdinov and Larochelle*, 2010). We leverage its most basic concept, vector quantization (*Theis et al.*, 2017; *Agustsson et al.*, 2017; *Oord et al.*, 2016), and use the discrete codes as our *IRs* in Neural Shape Compiler. We did not exhaust the complex variants or techniques for generality, such as Gumbel-softmax (*Jang et al.*, 2016; *Ramesh et al.*, 2021), the exponential moving average in codebook updating, and multi-scale structures (*Razavi et al.*, 2019; *Vahdat and Kautz*, 2020). Prior works studied learning 3D shape probabilistic spaces with GANs for shape synthesis (*Wu et al.*, 2016; *Nguyen-Phuoc et al.*, 2019); (*Achlioptas et al.*, 2018; *Yang et al.*, 2019; *Cai et al.*, 2020) learn the continuous distribution of point clouds and sample thereon to generate shapes; (*Mittal et al.*, 2022; *Cheng et al.*, 2022; *Yan et al.*, 2022) concurrently developed ways to quantize point clouds and voxels; diffusion models are also exploited in 3D shapes (*Luo and Hu*, 2021; *Zhou et al.*, 2021).

**3D Shape Understanding:** Our approach is related to research on 3D shape understanding about geometry processing and structural relationship discovery. Recent learning systems for shape processing usually perform over some low-level geometry representations, such as point clouds

(*Bronstein and Kokkinos*, 2010; *Qi et al.*, 2017b; *Liu et al.*, 2019; *Luo et al.*, 2020), volumetric grids (*Wu et al.*, 2015; *Maturana and Scherer*, 2015; *Wu et al.*, 2016; *Wang et al.*, 2019), multi-view images (*Bai et al.*, 2016; *Feng et al.*, 2018; *Han et al.*, 2019), meshes (*Groueix et al.*, 1802; *Lahav and Tal*, 2020; *Hu et al.*, 2022b), and implicit functions (*Mescheder et al.*, 2019; *Park et al.*, 2019; *Genova et al.*, 2020). Some represent shapes in a more structured way, including hierarchies, trees, and graphs (*Wang et al.*, 2011; *Li et al.*, 2017; *Sharma et al.*, 2018a; *Mo et al.*, 2019a). Neural Shape Compiler adopts point cloud as one of its shape abstractions because point cloud can describe shape structures and can be easily acquired by real sensors. Regarding the types of shape regularities (*Mitra et al.*, 2006; *Pauly et al.*, 2008; *Mitra et al.*, 2007; *Wang et al.*, 2011; *Xu et al.*, 2012), this paper mainly considers extrinsic part-level symmetries, including transformational, reflectional, and rotational symmetries, which relate to part pairs and multiple parts. Our framework adopts shape program (*Tian et al.*, 2019; *Jones et al.*, 2020, 2021; *Cascaval et al.*, 2022), which is one of the recent advances in relationship discovery. It designed a shape domain-specific language to encode shape relationships implicitly in programs.

## 2.3 Method

This paper aims to build a unified framework that can transform between different shape abstractions and benefit from joint multimodal learning. The key is tuning different shape abstractions into a unified space and learning thereon, as shown in Figure 2.2. Each type of input shape (*Point Cloud*, *Text*, or *Program*) is converted to unified discrete shape code ($[c_1, ..., c_n], \{c_i\} \in \mathbb{Z}+$) through the corresponding encoder. ShapeCode Transformer then transforms each shape code into the code that can be decoded into target shape abstraction via the corresponding decoder. Each type of encoder and decoder is designated to consume the specific type of shape abstraction, while ShapeCode Transformer communicates with all encoders and decoders via discrete code and handles transformations (*Text* $\Longrightarrow$ *Point Cloud* $\Longrightarrow$ *Program*).

### 2.3.1 Encoders & Decoders

To turn point clouds $P = \{p_1, \cdots, p_m\}$ ($p_i = (x_i, y_i, z_i)$) into codes $\mathscr{C}^{point} = ([c_1^p, c_2^p, ..., c_{N_{point}}^p]$, $\{c_i^p\} \in \mathbb{Z}+)$ while accommodating complex variations of shape structures, we take inspirations from VQVAE (*Van Den Oord et al.*, 2017) and propose *Point*VQVAE to conduct this task in a class-agnostic way. We design a new encoder and decoder to let *Point*VQVAE successfully reconstructs point clouds from shape code, the same representation that we turn text and shape programs into.

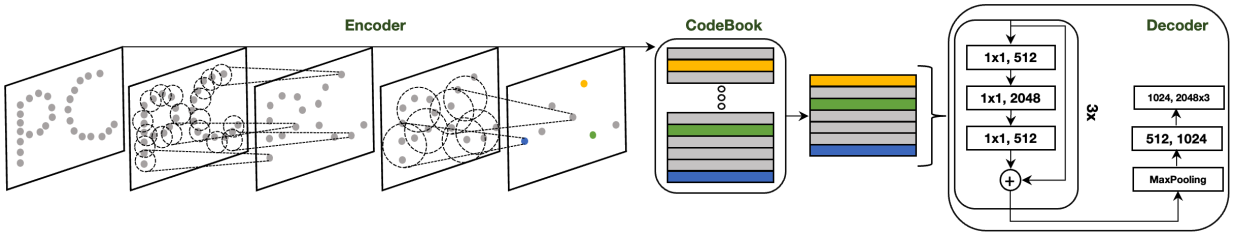



Figure 2.3: *Point*VQVAE consists of three parts: (1) a hierarchical encoder that has restricted receptive fields and outputs multiple part embeddings; (2) a shared codebook for vector quantization (*Van Den Oord et al.*, 2017) (e.g., looking up the closest yellow embedding in the codebook for the yellow output of the encoder); (3) a decoder with multiple residual building blocks consisting of 1x1 convolutional layers, a max-pooling layer, and MLP layers at the end to output 2,048 points.

The key point of the encoder is to constrain its receptive fields (*Luo et al.*, 2016) of the last layer neurons to very local regions (*Luo et al.*, 2019), and thus, let the codebook that follows learn how to encode parts of 3D shapes instead of entire 3D shapes (*Luo et al.*, 2020). To achieve this, we propose a shallow hierarchical encoder, implemented as a PointNet++-like structure (*Qi et al.*, 2017b) without losing generality. In each turn $T_i$, we first do farthest point sampling over input point clouds to sample a set of points noted as center points. Around each center point, we draw a ball with a radius $R_{T_i}$ to gather information from all points within the query ball. After this, the input point clouds are downsampled into smaller point clouds with updated embeddings on each point. Since the radius $R_{T_i}$ of our query ball is usually smaller than the scale of the whole point clouds, the receptive field of points in downsampled point clouds can be gradually increased as we repeatedly the downsampling process (i.e., increase $\#T$). In our experiments, we set the number of rounds $\#T = 2$ and $R_{T_i}$ to be small values ($R_{T_1} = 0.1, R_{T_2} = 0.4$) for letting the final neurons (e.g., colorful points of the encoder's last layer in Figure 2.3) only receive local part information regarding the

input shapes. Our experiments in Appendix A.3 demonstrate that making sure final neurons receive local information is crucial for reconstruction.

Similar to *Van Den Oord et al.* (2017), for each embedding in the last layer of our encoder, we look up its closest embedding in our codebook. We concatenate all the obtained embedding together as the input to our decoder. The concatenated embedding encodes the full information of the input shape by collecting the information encoded by different local parts of the input shape. Compared to common point clouds auto-encoders (*Qi et al.*, 2017b), we discard the position / coordinate information of the final points of the encoder. Not using positional information may degrade auto-encoding performance but allow us to translate between codes successfully where we do not have any positional information within the source codes (e.g., text and program). To compensate for this point, we designed a residual-network-like block (*He et al.*, 2016) and made our decoder with great depth to exploit the information from the concatenate embedding fully. The detailed structure of our decoder is in Figure 2.3. A max-pool layer is added to the decoder's output to ensure the final embedding is invariant to the concatenation order of the embedding (*Qi et al.*, 2017a). In our experiments, we adopt Chamfer distance (*Fan et al.*, 2017) and Earth Mover's distance (*Rubner et al.*, 2000) as our reconstruction loss and the straight-through gradient estimator (*Bengio et al.*, 2013) for backpropagating gradients to the encoder by copying the gradients from the inputs of the decoder to the encoder output. We followed the codebook objectives used in (*Van Den Oord et al.*, 2017) to update our codebook by moving our embedding vectors closer to the encoder outputs and preventing our embedding from growing too big. More details are included in Appendix A.2.1.

We adopt the simplest way to encode and decode text and programs without any external pretrained parameters. For text, we leverage BPE (*Sennrich et al.*, 2015) to encode and decode lowercase text. For program, due to the domain-specific language of shape program (*Tian et al.*, 2019) has limited discrete types of statements $\{A, B, C, \cdots\}$ and ranges of parameters for each statement $\{a_1, a_2, \cdots, b_1 \cdots\}$, We convert the shape program into code, first the statement type, then its parameters, i.e., $[A, a_1, a_2, a_3, B, b_1, b_2, b_3]$. For example, we encode the statement *draw('Top',*

*'Square', P=(-1,0,0), G=(2,5))* in Figure 2.2 as $[3,-1,0,0,2,5,0,0]$ where 3 represents the command of drawing square top and the remains are its specific parameters. This way provides a perfect precision to encode and decode shape programs while it cannot handle continuous parameters. Therefore, we proposed an extra ***Program*VQVAE** in Appendix A.2.3 to handle the case of the continuous parameters.

### 2.3.2 ShapeCode Transformer

After developing all the encoders and decoders, we can turn Point Clouds, Text, and Programs into the unified shape code $\mathscr{C}^{point}$, $\mathscr{C}^{text}$, and $\mathscr{C}^{program}$, respectively. ShapeCode Transformer performs over a pair of discrete codes and transforms from one type of code to another type. Similar to (*Ramesh et al.*, 2021), we model the pair of discrete codes as a single data stream. For example, if transforming to Point Cloud code from Text code, we model data like $[c^t_1,\cdots,c^t_{N_{text}},c^p_1,\cdots,c^p_{N_{point}}]$. In training, we pad 0 at the left of data (e.g., $[0,c^t_1,\cdots,c^t_{N_{text}},\cdots,c^p_{N_{point}}]$) and use full attention masks (*Child et al.*, 2019) to force ShapeCode Transformer autoregressively predicts the next token based on all the seen ones, i.e., predict $c_k$ with the context of $[0,\cdots,c_{k-1}]$. Assume $\{c^{gt}_k\}$ is the corresponding ground-truth token, our loss for this transformation is $\arg\min_\theta \sum_{k=1}^{N_{text}+N_{point}} \mathscr{L}(\mathscr{T}_\theta(0,c_1,\cdots,c_{k-1}),c^{gt}_k)$, where $\mathscr{L}$ is cross-entropy loss and $\mathscr{T}$ is ShapeCode Transformer parameterized by $\theta$ to predict $c_k$ by feeding $[0,\cdots,c_{k-1}]$.

As modeled above, the second type of code gets full attention to the first type of code in predictions. This modeling can be viewed as maximizing the evidence lower bound on the joint likelihood of the distribution over the input two types of code $p_\theta(\mathscr{C}^{type_i},\mathscr{C}^{type_j}), type \in \{point, text, program\}$, where $p_\theta$ is our ShapeCode Transformer. However, one of the challenges in our modeling is that we can generate either texts or programs conditional on the same input point clouds. *How do we let the ShapeCode Transformer know which type of object code we need?* Our solution is to encode our pairs of tokens with different positional embedding, i.e., we have two different positional encoding $\mathscr{P}_{\phi^{text}_{point}}([\mathscr{C}^{point},\mathscr{C}^{text}])$ and $\mathscr{P}_{\phi^{program}_{point}}([\mathscr{C}^{point},\mathscr{C}^{program}])$. Using a different positional encoding helps guide the ShapeCode Transformer to generate the specified target code.

$$\arg\min_{\theta,\phi,\omega} \sum_{k} \mathscr{L}(\mathscr{T}_{\theta}([\mathscr{P}_{\phi}(0), \mathscr{E}_{\omega}(0)], [\mathscr{P}_{\phi}(c_1), \mathscr{E}_{\omega}(c_1)], \cdots, [\mathscr{P}_{\phi}(c_{k-1}), \mathscr{E}_{\omega}(c_{k-1})]), c_k^{gt}) \quad (2.1)$$

ShapeCode Transformer communicates only the shape code with all the encoders and decoders. It maintains $\mathscr{E}_{\omega}(c) = [\#tokens, \#dim]$ embedding for each type of code $\{\mathscr{C}^{point}, \mathscr{C}^{text}, \mathscr{C}^{program}\}$ that will be optimized during training. In each optimization, once the code is received from the encoder, ShapeCode Transformer indexes the self-maintained embedding through the received code. It transforms shape code based on the indexed self-maintained embedding, enabling optimization without interference from encoder and decoder parameters. Our total training losses are the summation of equation 2.1 over multiple shape code pairs $c^i, c^j$ from $(\mathscr{C}^{text}, \mathscr{C}^{point})$, $(\mathscr{C}^{point}, \mathscr{C}^{text})$, and $(\mathscr{C}^{point}, \mathscr{C}^{program})$. For each input shape code, we concatenate its positional encoding $\mathscr{P}_{\phi}(c)$ and the embedding maintained by ShapeCode Transformer $\mathscr{E}_{\phi}(c)$ and feed into the ShapeCode Transfromer to predict the next token. In inference, we leverage Gumble noise (*Jang et al.*, 2016) to generate a set of plausible results for a given input. Please refer to A.2.2 for all the detailed parameters.

## 2.4 Experiments

We conduct *Text* $\Longrightarrow$ *Point Cloud*, *Point Cloud* $\Longrightarrow$ *Text* and *Point Cloud* $\Longrightarrow$ *Program* tasks to verify our performance, and *Partial Point Cloud* $\Longrightarrow$ *Complete Point Cloud* (Shape completion) task to show the extensibility of our framework (Appendix 2.4.4). For each task, we have two versions of our method: Shape Compiler Limited and Shape Compiler. **Shape Compiler Limited** means that we restrict both *Point*VQVAE and ShapeCode Transformer to train on the same data and task as the baselines for fair comparisons. **Shape Compiler** is our full model trained on all tasks and data to check if joint training on heterogeneous data and tasks leads to improvement. Our used data are briefly described below. More data collection details are listed in Appendix A.1.

**Shape-Text Pairs:** To investigate the structural connections between text and shape, we collect

a total of 107,371 (*Point Cloud*, *Structure-Related Text*) pairs from a total of 20,355 shapes with 9.47 words per description on average and 26,776 unique words. We use 85% shapes for training and the remaining 15% for testing. Data collection details are listed in Appendix A.1.

**3D Shape Assets**: To achieve a general framework, we collect various 3D shapes and obtain a total of 140,419 shapes of 144 categories from ShapeNet (*Chang et al.*, 2015), ABO (*Collins et al.*, 2021), and Program Synthetic (*Tian et al.*, 2019) datasets. For each shape, we sample 10,000 points as inputs and remove all artificial colors and textures to focus on shape geometry and compositionality.

**Shape-Program Pairs**: We followed (*Tian et al.*, 2019) and synthesized 120,000 (*Point Cloud*, *Program*) pairs. For testing, *Tian et al.* (2019) sampled shapes from ShapeNet, and we use the same sets to evaluate in our experiments for fair comparisons.

### 2.4.1 *Text* $\Longrightarrow$ *Point Cloud*

Neural Shape Compiler can generate point clouds corresponding to text prompt (Figure 2.4). We compare methods (*Jain et al.*, 2021; *Sanghi et al.*, 2022) that rely primarily on CLIP *Radford et al.* (2021b) to generate 3D shapes and methods that require 3D-text pairs (*Chen et al.*, 2018; *Liu et al.*, 2022b). We use our shape-text training pairs to train CWGAN (*Chen et al.*, 2018) and Shape IMLE diversified model (*Liu et al.*, 2022b), and our Shape Compiler Limited for fair comparisons. Since this paper focuses on geometry, we only compare with (*Chen et al.*, 2018; *Liu et al.*, 2022b) in geometrical aspects, which is the same comparison way used in (*Mittal et al.*, 2022). We compare CLIP-Forge (*Sanghi et al.*, 2022) and DreamField (*Jain et al.*, 2021) to show the gap between exploiting the 2D-text model and training over 3D-text pairs. Since DreamField learns a neural radiance field over a single text with tons of sampling, its training time is too long to test in the scale of our test set (details in Appendix A.2.4). We only compare its qualitative results. In order to benchmark *Text* $\Longrightarrow$ *Point Cloud* task, we turn models which output 3D voxels into point clouds by sampling 2,048 points on their output voxels. Shape Compiler is trained with all heterogeneous data and tasks described in Section 2.4 to show the benefits from jointly training over all tasks.

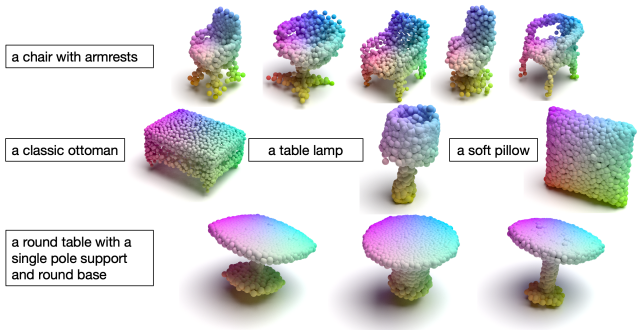



Figure 2.4: *Text* $\Longrightarrow$ *Point Cloud* by Neural Shape Compiler. A text prompt may translate to multiple shapes.

**Evaluation Protocol:** To benchmark *Text* $\Longrightarrow$ *Point Cloud* task, we measure generative performance from multiple angles: quality, fidelity, and diversity. **(1)** For measuring quality, we use Minimal Matching Distance (MMD) (*Achlioptas et al.*, 2018) to check if the generated shape distributions are close to ground-truth shape distributions by computing the distance between the set of all our generated point clouds and the set of all ground-truth point clouds. We use Chamfer distance for each pair of point clouds in computing MMD. **(2)** For measuring fidelity, we compute the minimal Chamfer distance between all generated point clouds and the corresponding ground truth shape of the input text. We report the Average Minimal Distance (AMD) over all the test texts. **(3)** For measuring diversity, we adopt Total Mutual Difference (TMD) (*Wu et al.*, 2020), which computes the difference between all the generated point clouds of the same text inputs. For each generated point cloud $P_i$, we compute its average Chamfer distance $CD_{P_i}$ to other $k-1$ generated point clouds $P_{j\,j\neq i}$ and compute the average: $TMD = \text{Avg}_{i=1}^{k} CD_{P_i}$. Given an input text, every model finalizes 48 normalized point clouds of 2,048 points.

**Results:** According to Table 2.1, Shape Compiler variants generally outperform CLIP-Forge (*Sanghi et al.*, 2022), CWGAN (*Chen et al.*, 2018), and Shape IMLE Diversified (*Liu et al.*, 2022b), especially in AMD (fidelity), which demonstrates our proposed method can generate shapes that more closely match the input texts which contain levels of geometrical details. This is consistent with the qualitative results in Figure 2.5, where all compared baselines cannot handle the text prompt containing many structural details well. One potential drawback in (*Chen et al.*, 2018; *Liu et al.*, 2022b) is the use of 3D convolutional layers, which make their models tend to overfit training data distribution. This is also shown in Figure 2.5, where their results are less correlated with the input texts. Shape IMLE outperforms CWGAN, consistent with the discovery in (*Liu et al.*, 2022b).

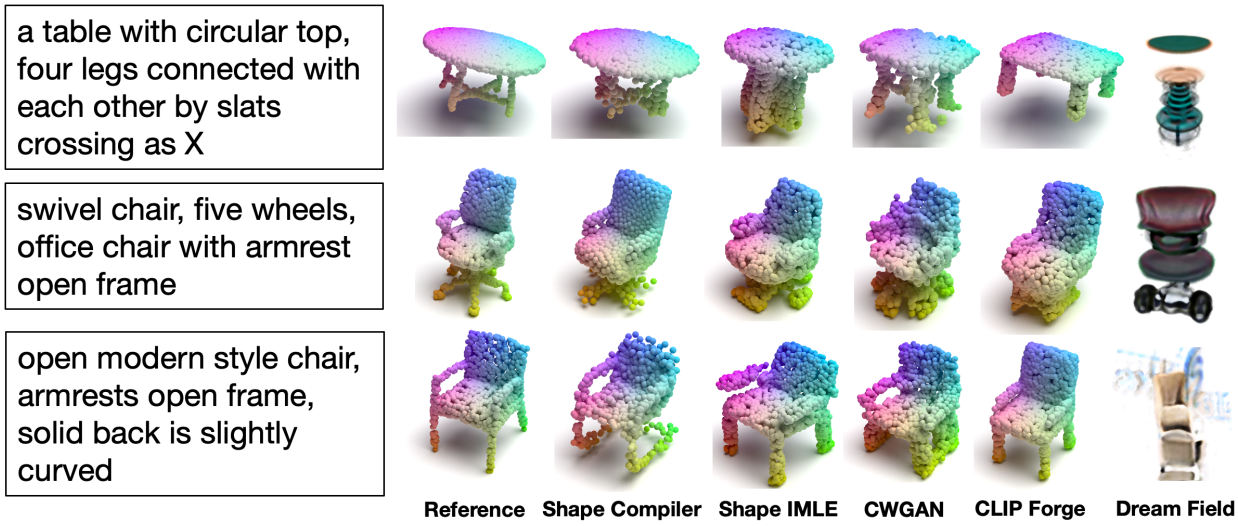



Figure 2.5: *Text* $\Longrightarrow$ *Point Cloud*. Neural Shape Compiler can generate corresponding point clouds to the text prompt with structural details, while the baselines generate inaccurate structures or fewer alignments with the text prompt.

However, its inference is much slower than Shape Compiler due to the use of 3D convolutional layers and implementations, whereas our method infers low-dimensional codes and decodes them through a 1D residual decoder. In our same single GPU, Shape IMLE takes 1003.23 seconds to generate 48 shapes, while Shape Compiler takes only 14.69 seconds. Shape Compiler consistently outperforms Shape Compiler Limited indicating the proposed framework benefits from the joint training on all heterogeneous data and tasks. However, due to the use of point cloud representation, the inherent difficulty of 3D shape generation with structure details, and the introduction of Gumbel noise, our framework produces rough boundaries, uneven surfaces, and irrelevant shapes. There is a lot of room for research in the proposed framework and *Text* $\Longrightarrow$ *Point Cloud* task.

Despite the success of using CLIP in 3D shape generation with text prompt containing simple structure descriptions (*Sanghi et al.*, 2022; *Jain et al.*, 2021), CLIP-Forge shows undesirable AMD and MMD scores in both ShapeGlot and ABO datasets, and DreamField fails to generate structures corresponding to the text prompt, although they are perceptually fine. Additionally, our experimental results suggest that **the text embedding of CLIP models (*Radford et al.*, 2021b) is *Not* able to reflect differences in small changes in the text, which may lead to large changes in structure**. Table 2.2 shows several examples. The texts in the second column are similar to those in the first but

| Dataset | Method | MMD ↓ | AMD ↓ | TMD ↑ |
|---|---|---|---|---|
| ShapeGlot | CWGAN | 22.46 | 27.32 | 0.67 |
| | Shape IMLE | 12.37 | 14.92 | 1.83 |
| | CLIP-Forge | 36.43 | 64.5 | 2.45 |
| | Shape Compiler Limited | 6.19 | 11.27 | 1.85 |
| | Shape Compiler | 5.7 | 7.31 | 2.8 |
| Text2shape | CWGAN | 10.42 | 18.21 | 0.79 |
| | Shape IMLE | 6.73 | 15.34 | 2.34 |
| | CLIP-Forge | 5.16 | 19.69 | 1.34 |
| | Shape Compiler Limited | 6.19 | 14.21 | 1.53 |
| | Shape Compiler | 4.53 | 11.66 | 3.07 |
| ABO | CWGAN | 12.74 | 22.31 | 0.52 |
| | Shape IMLE | 6.43 | 13.67 | 1.42 |
| | CLIP-Forge | 7.89 | 23.35 | 1.23 |
| | Shape Compiler Limited | 4.93 | 8.33 | 0.67 |
| | Shape Compiler | 4.76 | 7.77 | 1.34 |

Table 2.1: *Text* $\Longrightarrow$ *Point Cloud*. MMD (quality), AMD (fidelity), and TMD (diversity) are multiplied by $10^3$, $10^3$ and $10^2$, respectively.

will lead to significant structural changes. However, they have very high similarity scores in CLIP text embeddings, almost 1, which is an upper bound for cosine similarity. This may illustrate why CLIP-Forge (*Sanghi et al.*, 2022) achieved low AMD scores (Table 2.1), where our 3D-text datasets contain complex structural descriptions as shown in Appendix A.1. CLIP text embeddings cannot distinguish text with structural details well, which hurts the performance of text-guided 3D shape generation. The clip model we used here is *ViT-B/32*, and other CLIP models show similar trends.

Also, we found **the cosine similarity between the CLIP embedding of rendered 3D shape images and the CLIP embedding of text cannot accurately measure the degree of alignment between the text prompt and 3D geometry when the text prompt and 3D geometry contain structural details**. We render 3D shapes into images with six camera positions and two different point lights with PyTorch3D (*Ravi et al.*, 2020), as shown in Figure 2.6. Then, we obtain each image embedding by using the CLIP image branch encoder. Each shape has several text descriptions. We tokenize those texts with the tokenizer provided by CLIP and obtain our text embedding by feeding them into the CLIP text encoder. In Figure 2.6, the score above each shape is the average cosine

| Text1 | Text2 | Similarity |
|---|---|---|
| a chair with armrests | a chair with no armrests | 0.983 |
| a chair with armrests | a chair without armrests | 0.988 |
| a swivel chair with armrests | a swivel chair with no armrests | 0.979 |
| a swivel chair with armrests | a swivel chair without armrests | 0.986 |
| a chair with armrests | a chair with armrests and curved back | 0.962 |
| a chair with armrests | a chair with no armrests and curved back | 0.965 |
| a swivel chair with armrests | a swivel chair with armrests and curved back | 0.981 |
| a swivel chair with armrests | a swivel chair with no armrests and curved back | 0.982 |
| a table with circular top | a table with square top | 0.932 |
| a table with circular top | a table with rectangular top | 0.938 |
| a stool chair with two legs | a stool chair with three legs | 0.992 |
| a couch with two seats | a couch with three seats | 0.986 |

Table 2.2: The rightmost column is the cosine similarity between the CLIP text embeddings (*Radford et al.*, 2021b) of the left two texts. Cosine Similarity $= \frac{f_1 \cdot f_2}{\max(\|f_1\|_2 \cdot \|f_2\|_2, \varepsilon)}$, where $f_1$ and $f_2$ are the CLIP embedding of Text1 and Text2, respectively. $\varepsilon$ is 1e-6 to avoid division by zero.

similarity between the image and each text shown at the top.

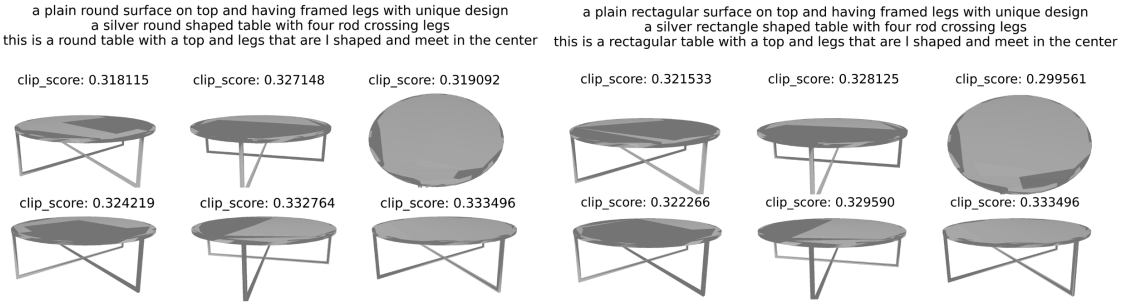



Figure 2.6: Render the shape with six different camera positions and two different point lights by Pytorch3D *Ravi et al.* (2020). The texts shown at the top left are the shape-associated texts provided in our datasets, which only contain Round or Circular. We modified them into Rectangle or Rectangular as shown in the top right. The number above each image is the average cosine similarity between each CLIP image embedding and all corresponding CLIP text embedding.

We select shapes with text containing Round or Circular and not including Rectangle and Rectangular. We compute the cosine similarity between the embedding of the rendered images and two different sets of text embedding: **(1)** the original ground-truth texts which contain Round or Circular; **(2)** we replace Round or Circular into Rectangle. An example of replacement is shown in Figure 2.6 where the right figure top texts are the replaced ones. We compute the similarity scores over 1,358 shapes with a total of 6,697 texts. As described above, we have six images for each shape,

resulting in 40,182 scores. We calculate the mean and standard deviation of all scores for the two sets of text: (1) $0.2925 \pm 0.02173$ and (2) $0.2917 \pm 0.02211$. Although there are huge differences between the two sets of texts, their average cosine similarity with the CLIP image embedding is very close. This experiment further supports CLIP is *NOT* suitable to deal with text containing geometric details.

### 2.4.2 *Point Cloud* $\Longrightarrow$ *Text*

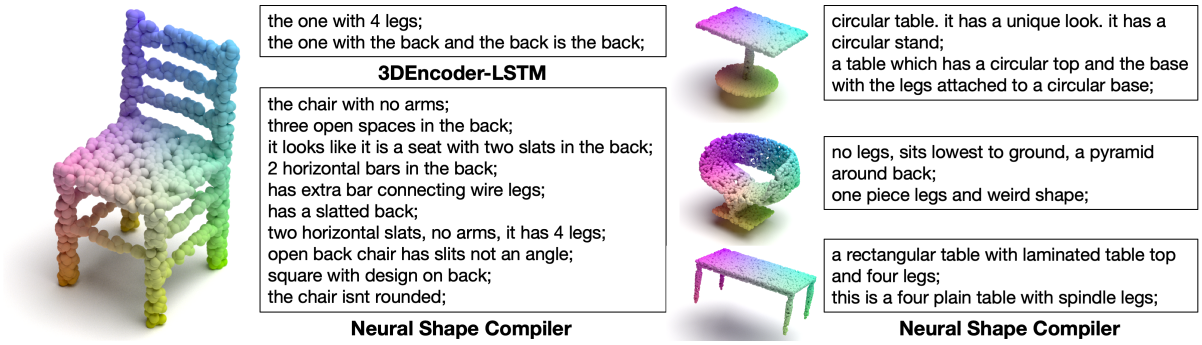



Figure 2.7: *Point Cloud* $\Longrightarrow$ *Text*. One description per sentence. The shown shapes are from test sets, and Neural Shape Compiler tell their structures well.

Neural Shape Compiler performs *Point Cloud* $\Longrightarrow$ *Text* tasks that describe given shape point clouds as shown in Figure 2.7. To compare, we followed (*Chen et al.*, 2021a) and implemented a 3DEncoder-LSTM model as our baseline by replacing the image encoder of Show-Attend-Tell(*Vinyals et al.*, 2015b) with a 3D encoder. We adopt a similar encoder structure as *Point*VQVAE and set the output feature to be 512 dimension, the same value as our intermediate representations. An LSTM model will then be trained over the 512 output features to predict words gradually, and an end word will be the last token. For each shape, all methods will generate 48 descriptions to compare.

**Evaluation Protocol:** We measure the quality of the generated shape descriptions with our annotations in validation sets. Here we adopt the common metrics used in text generations tasks (*Vinyals et al.*, 2015b; *Koncel-Kedziorski et al.*, 2019) including BLEU (4-gram) (*Papineni et al.*, 2002), CIDER (*Vedantam et al.*, 2015), and ROUGE (*Lin*, 2004). We also adopt Dist-1 and Dist-2

| Dataset | Method | BLEU-4 ↑ | CIDER ↑ | ROUGE ↑ | Dist-1 ↑ | Dist-2 ↑ |
|---|---|---|---|---|---|---|
| ShapeGlot | 3DEncoder-LSTM | 2.78 | 2.93 | 18.61 | 0.016 | 0.017 |
| | Shape Compiler Limited | 2.96 | 2.15 | 22.26 | 1.174 | 14.146 |
| | Shape Compiler | 3.89 | 3.35 | 22.78 | 1.698 | 17.764 |
| | Ground-Truth | - | - | - | 6.908 | 37.92 |
| Text2Shape | 3DEncoder-LSTM | 1.01 | 0.84 | 21.18 | 0.027 | 0.042 |
| | Shape Compiler Limited | 2.96 | 1.5 | 25.97 | 0.792 | 9.984 |
| | Shape Compiler | 3.02 | 1.52 | 25.52 | 0.828 | 10.109 |
| | Ground-Truth | - | - | - | 5.906 | 34.05 |
| ABO | 3DEncoder-LSTM | 0.01 | 1.39 | 2.98 | 0.031 | 0.047 |
| | Shape Compiler Limited | 8.95 | 111.28 | 26.89 | 3.605 | 5.082 |
| | Shape Compiler | 10.03 | 127.82 | 28.98 | 3.968 | 24.176 |
| | Ground-Truth | - | - | - | 18.294 | 49.097 |

Table 2.3: *Point Cloud* $\Longrightarrow$ *Text*. BLEU, CIDER, and ROUGE measure quality. Dist-1 and Dist-2 measure diversity.

in (*Li et al.*, 2015) to measure the diversity of the generated results: Dist-1 and Dist-2 are the results of distinct unigrams and bigrams divided by the total number of tokens (maximum 100 in our table).

**Results:** According to Dist-1 and Dist-2 in Table 2.3, Shape Compiler generates much more diverse captions than 3DEncoder-LSTM (*Vinyals et al.*, 2015b; *Chen et al.*, 2021a) due to the help of involving Gumbel noise over the learned priors in the inference. This is also shown in Figure 2.7 where Shape Compiler can describe given shapes in multi-angles. Also, our framework achieves generally higher BLEU, CIDER, and ROUGE scores than 3DEncoder-LSTM. By inspecting the predictions, we found that 3DEncoder-LSTM tends to predict the frequent words in datasets (e.g., back and legs), and the output descriptions are less meaningful in semantics. However, those repetitive words can help them achieve higher BLEU, CIDER and ROUGE scores in Text2Shape and ShapeGlot. Therefore, evaluating *Point Cloud* $\Longrightarrow$ *Text* performance with both quality and diversity metrics is essential. Furthermore, the ground-truth descriptions differ significantly from our predictions on diversity metrics, suggesting that the proposed method still has huge gaps in achieving human-level shape captioning.

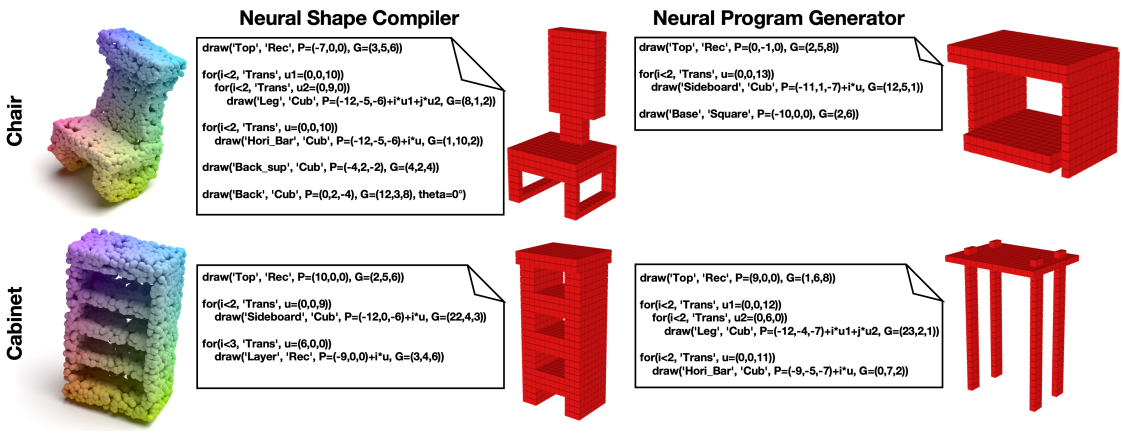



Figure 2.8: *Point Cloud* $\Longrightarrow$ *Program*. Neural Shape Compiler can well infer programs for the shown shapes of unseen categories and reconstruct shapes in a creative manner (e.g., represent the semicircle leg with bars).

### 2.4.3 *Point Cloud* $\Longrightarrow$ *Program*

Shape Compiler performs transformation: *Point Clouds* $\Rightarrow$ *Program*, which can help us understand how the point clouds are assembled by parts and regularities. For example, in Figure 2.8, we can tell how to decompose the point clouds into primitives via the symbolic words in the program and also find the mapping between points and the corresponding primitive via executing the program. Furthermore, the symmetry relationship between those racks of the cabinet is implicitly encoded in the *FOR* statements. The discovery of parts and relationships can help us design robots better interacting with 3D objects.

Neural Program Generator (*Tian et al.*, 2019) is our most direct baseline. They used a two-layered LSTM to gradually generate program blocks and programs inside each block to form the final shape of programs. We will not adopt the guided adaptation used in (*Tian et al.*, 2019) in our experiments because it needs extra training loops and prevents the practical use of program generation (further discussion is included in Appendix A.4). We also compare with CSGNet (*Sharma et al.*, 2018a), which applies boolean operations on shape primitives and assemble shape recursively.

**Evaluation Protocol:** Shape Program (*Tian et al.*, 2019), CSGNet (*Sharma et al.*, 2018a), and our models will predict programs under specific domain languages. By executing the output

| Metric | Method | Chair | Table | Bed | Sofa | Cabinet | Bench | Avg |
|---|---|---|---|---|---|---|---|---|
| IoU ↑ | CSGNet (*Sharma et al.*, 2018a) | 0.365 | 0.406 | - | - | - | - | - |
| | Program Generator (*Tian et al.*, 2019) | 0.438 | 0.517 | 0.254 | 0.324 | 0.304 | 0.216 | 0.342 |
| | Shape Compiler Limited | 0.429 | 0.539 | 0.27 | 0.31 | 0.504 | 0.314 | 0.394 |
| | Shape Compiler | 0.47 | 0.631 | 0.264 | 0.419 | 0.524 | 0.389 | 0.45 |
| CD ↓ | CSGNet (*Sharma et al.*, 2018a) | 7.73 | 7.24 | - | - | - | - | - |
| | Program Generator (*Tian et al.*, 2019) | 1.64 | 1.97 | 4.78 | 3.14 | 2.95 | 2.71 | 2.87 |
| | Shape Compiler Limited | 1.53 | 1.21 | 4.51 | 2.84 | 1.58 | 1.57 | 2.21 |
| | Shape Compiler | 1.27 | 0.92 | 4.39 | 2.52 | 1.49 | 1.62 | 2.04 |

Table 2.4: *Point Cloud* $\Longrightarrow$ *Program*. CD is multiplied by $10^2$. Avg denotes average number among categories.

programs, we can obtain volumetric representation shapes; then, we can compute IoU based on the output voxels. Also, we sample points over the surface of output voxels for computing Chamfer distance. Due to the benefits of being a probabilistic method, our framework can predict many programs given a single point cloud. We here generate 48 programs for comparisons, the same number we used in our text tasks. Ablation studies of the generated program numbers are included in Appendix A.4.

**Result:** According to Table 2.4, Shape Compiler outperforms all the methods due to the benefits of multimodal learning on all tasks and data. It also suggests our framework can effectively take advantage of task-unrelated heterogeneous data. Figure 2.8 shows two positive examples of our framework that better handles novel data than the baseline. Both cases show that the compared baseline mainly relies on memorizing the training table data, while Shape Compiler successfully predicts the rough structure and the regularities. Regarding the scores, Shape Compiler Limited achieved comparable performance with Shape Generator, while the significant performance gap in cabinet and bench categories shows our framework has stronger generalizability. Due to the limitation of the current program grammar, our method cannot handle complex structures for now. We raise attention to the next-level shape program, and some further discussion is included in Appendix A.4.

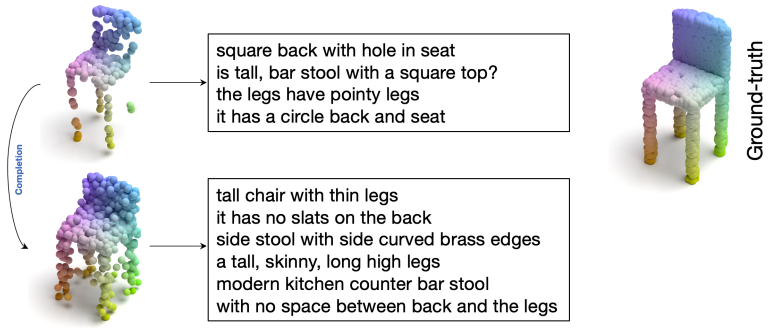



Figure 2.9: The success of shape completion boosts shape structural understandings. We use Neural Shape Compiler to do *Point Cloud* $\Longrightarrow$ *Text* for both the partial chair and the complete chair. The ground-truth shape is on the top right.

| Method | Chair | Airplane | Car |
|---|---|---|---|
| PointFlow (*Yang et al.*, 2019) | 6.93 | 1.07 | 3.97 |
| PVD (*Zhou et al.*, 2021) | 7.34 | 1.19 | 3.83 |
| Shape Compiler Limited | 7.57 | 1.62 | 4.82 |
| Shape Compiler | 6.9 | 1.56 | 3.81 |

Figure 2.10: *Partial Point Cloud* $\Longrightarrow$ *Complete Point Cloud*. Numbers are Chamfer distance $\downarrow$ with multiplying $10^2$.

### 2.4.4 *Partial Point Cloud* $\Longrightarrow$ *Complete Point Cloud*

In this section, we extend Neural Shape Compiler to conduct *Partial Point Cloud* $\Longrightarrow$ *Complete Point Cloud* (shape completion) task by projecting both partial and full point clouds into discrete codes and performing conditional generation thereon. We followed the benchmark provided in (*Zhang et al.*, 2018; *Zhou et al.*, 2021) and the detailed data preparation is listed in Appendix A.1.3. Different from the way used in *Zhou et al.* (2021), we train models on all training shapes in a class-agnostic way. Results in Table 2.10 show Shape Compiler achieved great performance compared with the previous method. Also, PVD (*Zhou et al.*, 2021) is slower than Shape Compiler because they involve very long steps (1,000) in the diffusion process.

Additionally, with the help of Neural Shape Compiler, we may better share the progress across different tasks. In Figure 2.9, we individually perform *Point Cloud* $\Longrightarrow$ *Text* over the partial chair and the complete chair. The ground-truth chair is on the right. For the partial chair, the captioning results include hallucinated descriptions, like *hole in the back* and *circle back*, and also accurate descriptions of the partial chair but not precisely related to the ground-truth one, like *pointy legs*. Compared to it, the descriptions for the completion result are more related to our ground-truth point cloud.

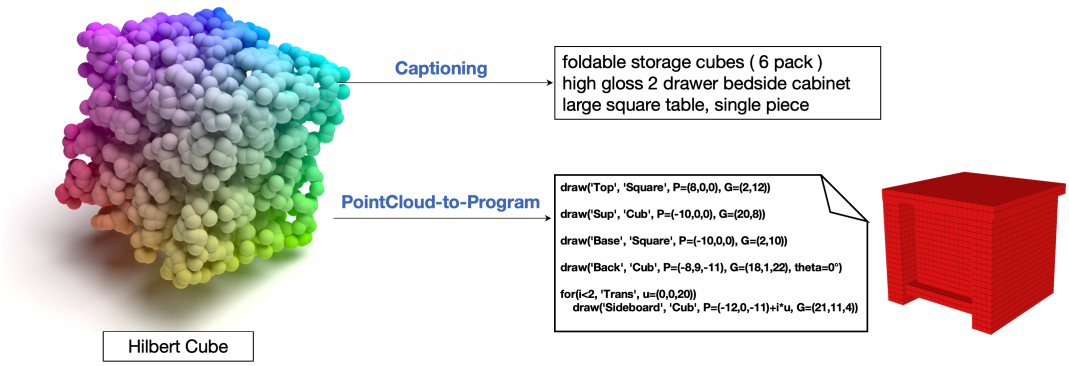



Figure 2.11: *Point Cloud* $\Longrightarrow$ *Text* and *Point Cloud* $\Longrightarrow$ *Program* results over the input Hilbert cube point clouds by Shape Compiler. One description per sentence.

## 2.5 Limitations and Future Works

Our current framework takes a step further in *Text* $\Longrightarrow$ *Point Cloud*, *Point Cloud* $\Longrightarrow$ *Text*, *Point Cloud* $\Longrightarrow$ *Program*, and Point Cloud Completion tasks. However, there are still a lot of things to be considered: 1. Similar to (*Chen et al.*, 2018; *Liu et al.*, 2022b; *Mittal et al.*, 2022), our method requires large-scale paired data, where the 3D-text pairs are hard to obtain. This situation could be alleviated by the development of large-scale 3D-text datasets and recovery of 3D data from 2D images *Gkioxari et al.* (2022) and videos *Qian et al.* (2022) that contain paired text information.

2. The proposed framework currently cannot well handle out-of-distribution geometry, such as the dragon in (*Morreale et al.*, 2022) and the Hilbert cube in (*Liu et al.*, 2020a). Figure 2.11 shows the results of *Point Cloud* $\Longrightarrow$ *Text* and *Point Cloud* $\Longrightarrow$ *Program* over the Hilbert cube by Shape Compiler. The current programs can only tell the coarse geometry, and captions are about similar objects in our training data, such as cubes and cabinets.

3. Our approach currently adopts the domain-specific language (DSL) proposed in (*Tian et al.*, 2019) as one of the input and target shape abstractions. However, the DSL's grammar and syntax are not powerful enough to handle various complex shapes. For example, it fails to perform accurate reconstruction or completion for 3D shapes with geometrical details (*Chaudhuri et al.*, 2020) and detect self-symmetries possessed by individual parts (*Mitra et al.*, 2006). Since the progress of shape programs is orthogonal to the proposed framework, the development of shape programs can further benefit our framework. Therefore, this paper also raises attention to developing the next

level of shape programs.

4. Our current framework and data focus on correctly modeling connections between 3D geometry and text. However, less attention has been paid to color, texture, and material, which will be investigated in the future with the support of appropriate data.

5. Our framework is extensible, where we showed a case in shape completion task (Table 2.10). Similarly, we can project images (*Van Den Oord et al.*, 2017), tactile information (*Gao et al.*, 2022b), and other modalities into the unified representation to obtain powerful representations for benefiting more downstream tasks, such as multiview reconstruction (*Ji et al.*, 2017) and multi-modal grasping (*Calandra et al.*, 2018).

6. DALL-E (*Ramesh et al.*, 2021) ranks their outputs of (*Text*, *Image*) based on the CLIP embedding similarity scores (*Radford et al.*, 2021b). However, as the experiments conducted in Section 2.4.1, the CLIP similarity score cannot properly reflect the alignment between 3D shape and text if the text contains geometric details. How to align 3D objects and text is a fundamental problem we should study in the future.

7. Neural Shape Compiler currently compiles different shape abstractions. Compared to this, a further direction is to compile operations on one shape abstraction into practical steps in other abstractions and change it accordingly. For example, we add "*armless*" into a text description of a chair with two arms. This change can be effectively compiled into operations "*remove two arm parts and their symmetry relationship*" in the shape hierarchy and "*delete all points belonging to the two arms*" in point clouds. We leave the shape editing direction to future study.

## 2.6 Conclusion

We proposed Neural Shape Compiler to transform between three shape abstractions: *Text*, *Point Cloud*, and *Program*. With the help of *Point*VQVAE, it achieved great performance in *Text* $\Longrightarrow$ *Point Cloud*, *Point Cloud* $\Longrightarrow$ *Text*, *Point Cloud* $\Longrightarrow$ *Program*, and PointCloud Completion tasks via a unified and extendable framework. Our experiments show that Neural Shape Compiler benefits from joint training on all heterogeneous data and tasks. Despite showing promises, Shape

Compiler has limitations and draws attention to several related directions for future research (Section 2.5). Furthermore, we study CLIP embeddings in text-guided shape generation and find that it is unsuitable for use once the text contains geometric details. We hope that Neural Shape Compiler can serve as an effective framework for connecting different shape abstractions, and the studies in this paper can facilitate the progress in 3D multimodal learning research.

# CHAPTER 3

# Scalable 3D Captioning with Pretrained Models

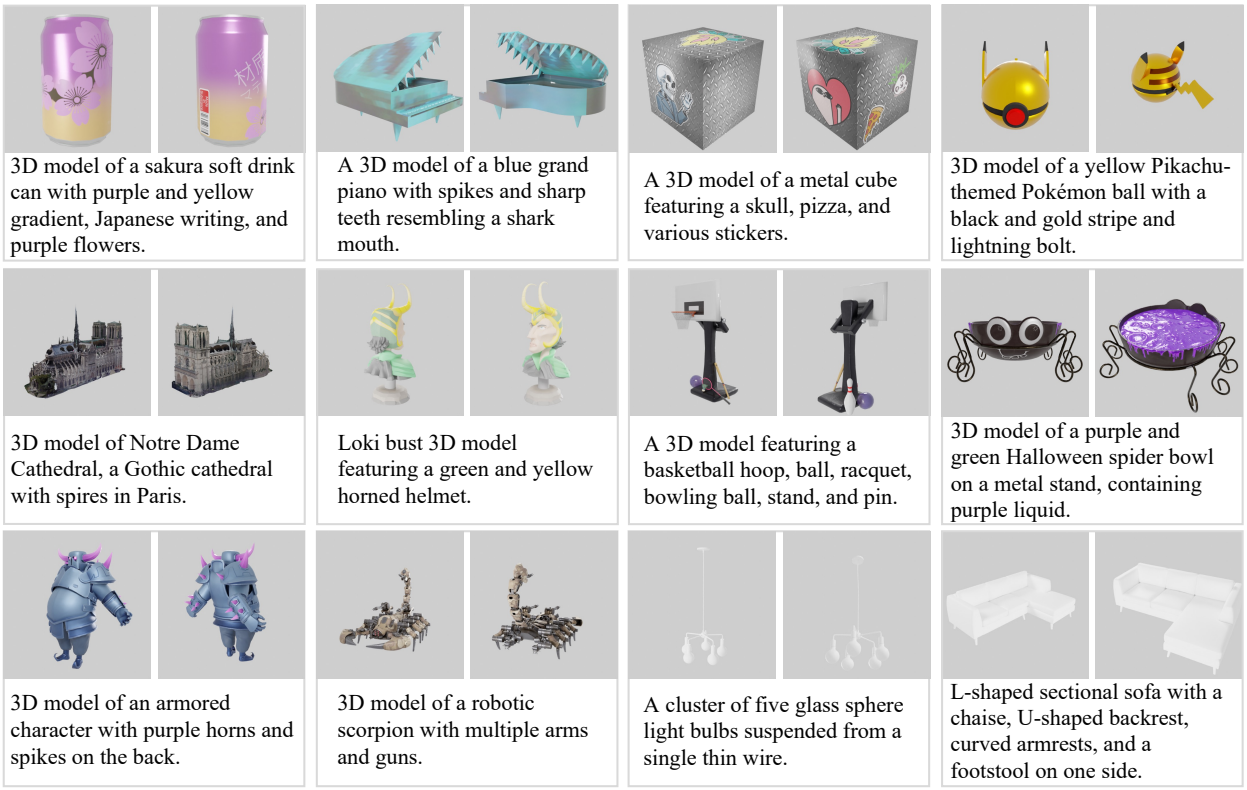



Figure 3.1: Cap3D provides detailed descriptions of 3D objects by leveraging pretrained models in captioning, alignment, and LLM to consolidate multi-view information. Two views of 3D objects are shown here, Cap3D uses eight. Additional examples are available in Appendix B.2.

This chapter is based on work published at NeurIPS 2023 (*Luo et al.*, 2023b).

*The previous chapter introduced a unified representation for translating between shape modalities. Yet learning such representations at scale requires supervision—specifically, large collections of 3D objects paired with informative text. This chapter addresses that need by developing* ***structure as supervision****: we exploit the multi-view geometry of 3D assets to construct high-quality captions automatically, turning pretrained 2D vision–language models into scalable annotators for the 3D domain.*

## 3.1 Introduction

Text-conditioned 3D synthesis *Jun and Nichol* (2023); *Gupta et al.* (2023); *Poole et al.* (2022) could revolutionize the creation process of 3D assets, impacting various sectors, including 3D design, virtual reality *Tang and Ho* (2020), film *Parent* (2012), robotics *Afzal et al.* (2020); *Li et al.* (2023c), and autonomous driving *Dosovitskiy et al.* (2017). However, challenges persist, namely the high cost of 3D asset creation and the scarcity of high-quality captions for 3D assets. Objaverse *Deitke et al.* (2023a) takes a step towards this as the first public large-scale 3D object dataset. Unfortunately, while objects contain paired metadata, these do not serve as informative captions, as shown in Table 3.3. In contrast with 3D, a plethora of high-quality text-image paired data is publicly available *Desai et al.* (2021); *Sharma et al.* (2018b); *Changpinyo et al.* (2021); *Schuhmann et al.* (2021, 2022). This data has led to incredible recent progress in image-text learning *El Banani et al.* (2023); *Radford et al.* (2021b); *Mu et al.* (2022); *Alayrac et al.* (2022b), text-conditioned image synthesis *Ramesh et al.* (2021); *Gafni et al.* (2022); *Saharia et al.* (2022); *Rombach et al.* (2022); *Nichol et al.* (2021); *Ramesh et al.* (2022), and image captioning *Wang et al.* (2022b); *Li et al.* (2020); *Zhang et al.* (2021a); *Han et al.* (2020); *Li et al.* (2023d).

In this work, we present Cap3D, a method to automate 3D object annotation. Our key insight is to leverage the abundance of knowledge in pretrained image-text models to remedy the lack of existing 3D-text data. The core of our data collection process is to apply an image captioning model (BLIP2 *Li et al.* (2023d)) to a set of 3D asset renders, use an image-text alignment model

(CLIP *Radford et al.* (2021b)) to filter captions, and apply a language model (GPT4 *OpenAI* (2023b)) to fuse the filtered captions across views. Critically, the models we apply are pretrained on varied and large-scale text-image *Lin et al.* (2014); *Krishna et al.* (2017); *Sharma et al.* (2018b); *Changpinyo et al.* (2021); *Ordonez et al.* (2011); *Schuhmann et al.* (2021), and text *Common Crawl*, data; and approach complementary problems. As a result, each model adds additional value to the framework, as we show in Table 3.3.

Cap3D is agnostic to 3D asset sources and can be effectively scaled to larger extents with increased 3D assets and computational resources. In this paper, we apply it primarily to Objaverse, gathering a dataset of 785k 3D-text pairs. Through object rendering and captioning, we enable ethical filtering of 3D objects via both image and text, as detailed in § 3.3.2. We publicly release all of our collected data including automated and human-annotated captions, along with associated Point Clouds and Rendered Images, at huggingface.co/datasets/tiange/Cap3D. The dataset is released under ODC-By 1.0 license. We also released trained models and code for replicating the benchmark table.

We validate our collection approach by collecting over 50k crowdsourced captions on over 40k objects. We conduct human evaluations and show on Objaverse that our automated captions are superior to crowdsourced captions in quality, cost, and speed (Table 3.1, details in Appendix B.1). Specifically, it is preferred 35% more often by humans, costs more than 10 times less, and is over 40 times faster, assuming only 8A40 GPUs. We also test the limits of automated captioning. We consider a separate task of captioning geometry (as shown in Figure 3.1 bottom-right) using ABO, a dataset of 3D models with complex geometries *Collins et al.* (2022). Shown in Table 3.4, our automated captioning underperforms humans. However, by formulating description as a question answering task (detailed in § 3.3.1), we show stronger performance compared to crowdsourced workers. This result shows the ability of our method to adapt beyond traditional captioning and still be highly competitive.

Finally, our high-quality gathered 3D-text dataset enables us to train and validate large-scale text-to-3D models. In §3.5.3, we evaluate several state-of-the-art methods on Objaverse out-of-the

Table 3.1: Cap3D is ***better, cheaper, and faster*** than crowdsourced annotation. Use 36k responses across 22k objects for A/B testing; 8A40s on a cloud platform for speed and cost computations.

| Method | A/B Human Testing Win % (Tie %) | Cost per 1k Objects | Annotation Speed |
|---|---|---|---|
| Human | 37.8% ± 0.5% (9.5%) | $87.18 | 1.4k / day |
| Cap3D | **52.3% ± 0.5% (9.5%)** | **$8.35** | **65k / day** |

| 1k Objects Cost Breakdown | |
|---|---|
| BLIP2 | $3.79 |
| CLIP | $0.38 |
| GPT4 | $4.18 |
| Cap3D Total Cost | **$8.35** |

box, including Point·E, Shap·E, DreamFields, and DreamFusion. Finetuning on our data typically shows meaningful improvements, demonstrating the value of the collected dataset. In addition, we show our automatically collected captions yield better finetuning performance than human captions – even at the same scale. At full scale, finetuning is further boosted.

## 3.2 Related Work

Obtaining 3D-text pairs at scale is challenging, and we take inspiration from image-text datasets and methods when approaching this task.

**Image-Text Data and Modeling** Early image captioning *Anderson et al.* (2018a); *Rennie et al.* (2017); *Lu et al.* (2018) and text-image representation learning methods *Yu et al.* (2018); *Li et al.* (2019a); *Lee et al.* (2018) were built using CNNs *He et al.* (2016); *Krizhevsky et al.* (2012); *Ronneberger et al.* (2015) and LSTMs *Hochreiter and Schmidhuber* (1997); *Schuster and Paliwal* (1997), leveraging human-annotated datasets *Ordonez et al.* (2011); *Lin et al.* (2014); *Krishna et al.* (2017); *Agrawal et al.* (2019). Text-to-image methods used similar datasets, and relied on GANs *Goodfellow et al.* (2014); *Karras et al.* (2020) and VQVAEs *Van Den Oord et al.* (2017); *Esser et al.* (2021); *Ramesh et al.* (2021); *Ding et al.* (2021). The advent of semi-automated image-text collection has enabled successful scaling of datasets *Desai et al.* (2021); *Sharma et al.* (2018b); *Changpinyo et al.* (2021); *Schuhmann et al.* (2021, 2022) and models *Wang et al.* (2022b); *Li et al.* (2020); *Zhang et al.* (2021a); *Han et al.* (2020). Transformer-based architectures *Desai and Johnson* (2021); *Radford et al.* (2021b); *Dosovitskiy et al.* (2021) and diffusion models *Dhariwal and Nichol* (2021); *Ho et al.* (2020); *Karras et al.* (2022); *Nichol and Dhariwal* (2021); *Song and Ermon* (2019);

*Sohl-Dickstein et al.* (2015) have scaled best to large data; we employ transformer-based methods through our captioning process and adopt diffusion models for text-to-3D experiments.

Training models upon large datasets and using the corresponding trained models to filter larger data has led to datasets of rapidly increasing size *Schuhmann et al.* (2021, 2022). In addition to filtering, trained models have been used to annotate new data with high-quality *Kirillov et al.* (2023). We take this approach, captioning rendered views with BLIP2 *Li et al.* (2023d), refining with CLIP *Hessel et al.* (2021); *Radford et al.* (2021b), and summarizing with GPT4 *OpenAI* (2023a); all of which are trained on large datasets, including *Lin et al.* (2014); *Krishna et al.* (2017); *Sharma et al.* (2018b); *Changpinyo et al.* (2021); *Ordonez et al.* (2011); *Schuhmann et al.* (2021). Concurrent works *Zhang and Agrawala* (2023); *Pinkney* (2022); *Liu et al.* (2023e) use automated captioning on 2D images using an older system *Li et al.* (2022) or based upon metadata *Xue et al.* (2023a); *Liu et al.* (2023e).

**3D-Text Data and Modeling** Until recently, 3D data was of relatively small scale ($\sim$ 50k objects) *Chang et al.* (2015); *Sun et al.* (2018); *Lim et al.* (2013); *Fu et al.* (2021). Labeled 3D-text data was scarce, relying on human annotation, and typically limited to ShapeNet *Chang et al.* (2015) chairs *Achlioptas et al.* (2019b) or tables and chairs *Chen et al.* (2019); *Fu et al.* (2022), and ScanNet *Dai et al.* (2017); *Chen et al.* (2020). This enabled prior work to undertake the task of 3D captioning *Luo et al.* (2023a); *Vinyals et al.* (2015a); *Chen et al.* (2021b) or text-to-3D *Chen et al.* (2019); *Luo et al.* (2023a); *Sanghi et al.* (2022); *Mittal et al.* (2022); *Wei et al.* (2023); *Zhang et al.* (2023a) at small scale. Methods that approached text-to-3D would sometimes avoid 3D supervision entirely *Jain et al.* (2022); *Poole et al.* (2022); *Tsalicoglou et al.* (2023); *Lin et al.* (2023), leading to slow generation due to many optimization steps. We annotate a small-scale dataset containing 3D furniture, ABO *Collins et al.* (2022), to evaluate the ability of Cap3D to specify fine-grained geometry.

Objaverse *Deitke et al.* (2023a) introduced a diverse set of objects over 10 times the size of the prior largest public 3D dataset *Chang et al.* (2015). This data is our primary captioning focus, and we associate a single caption with each object in Objaverse after filtering. Concurrent works *Xue*

*et al.* (2023b); *Liu et al.* (2023e) gather text associated with Objaverse, but do not fuse captions across views *Xue et al.* (2023b) or rely upon metadata *Liu et al.* (2023e), and do not approach text-to-3D.

The concurrent studies 3DGen *Gupta et al.* (2023) learns text and image to 3D on Objaverse; Point·E *Nichol et al.* (2022) and Shap·E *Nichol and Jun* (2023) learn text-to-3D models on a large-scale 3D dataset, but none have fully disclosed their code or data. Point·E involves two variants and released a text-to-3D model and a text-to-image-to-3D model by finetuning GLIDE *Nichol et al.* (2021) and training an image-to-point cloud diffusion model *Zhou et al.* (2021). Other recent works *Wu et al.* (2023); *Liu et al.* (2023g) also focus on scaled image-3D generation. We show finetuning on our captions improves Point·E performance despite having already been trained on large amounts of Internet data.

## 3.3 Method

### 3.3.1 Captioning Process

Our task is to produce a single descriptive caption given a 3D asset. Our proposed method, Cap3D, employs a four-step process. First, we render a set of 2D views for each 3D object. Next, we apply image captioning to achieve preliminary descriptions. As these captions may contain inaccuracies, an image-text alignment model, CLIP, is introduced in the third step to rectify errors. Finally, an LLM is employed to unify captions from various perspectives, creating a comprehensive caption. This process is shown in Figure 3.2 and detailed below.

**Object Rendering**: We render using Blender at $512\times512$ from $M = 8$ high-information camera angles rotating horizontally around the object, with two slightly below and the rest slightly above the object, to cover all the object details. The reason we prefer multiple views is a forward-facing view may miss self-occluded object details (e.g. Figure 3.1 row 1) or face strange appearance and/or lighting. In contrast, multiple views will see much of the object from different viewpoints, increasing the number of chances for a captioning model to predict objects in detail. For instance,

in Figure 3.2, the back view 1 identifies the "yellow handle", which is barely visible in forward view $M$.

**Image Captioning:** We use BLIP2 *Li et al.* (2023d) for captioning, selecting the largest pretrained model adapting ViT-G *Fang et al.* (2023); *Dosovitskiy et al.* (2021) image encoder and FlanT5XXL *Chung et al.* (2022) text encoder. We generate $N = 5$ captions per rendered image using nucleus sampling *Holtzman et al.* (2020). By generating multiple captions, we increase the likelihood of generating correct details (e.g. "black and yellow toy bomb" in Figure 3.2 view $M$ caption 1). Incorrect captions, such as "scissors" in Figure 3.2 view $M$ caption $N$, can be filtered in later stages. To generate captions containing fine-grained geometry details (in our ABO experiments), we employ a two-stage question-answering instead of captioning. The first stage generates one answer to a prompt asking what object is pictured. The answered object is passed into a second prompt, which asks its structure and geometry, and generates 5 answers.

**Caption Selection:** While BLIP2 often generates high-quality captions, it is not uncommon for samples to contain mistakes, particularly in non-forward facing views such as "yellow cup", in Figure 3.2 view 1, caption $N$. To reduce the frequency of mistakes, we compute CLIP *Radford et al.* (2021b) ViT-B/32 *Dosovitskiy et al.* (2021) encodings from each of 5 captions and the associated image, and select the caption maximizing cosine similarity. CLIP tends to select good captions for each view, e.g. Figure 3.2: view 1, BLIP2 caption 1 and view $M$, caption 1. CLIP is complementary to BLIP2 as not only does it have different training details and architecture, but it trains on different data. While BLIP2 is trained upon COCO *Lin et al.* (2014), Visual Genome *Krishna et al.* (2017), CC3M *Sharma et al.* (2018b), CC12M *Changpinyo et al.* (2021), SBU *Ordonez et al.* (2011) and LAION400M *Schuhmann et al.* (2021); CLIP is trained upon a dataset of 400M images based on frequent text occurrence in Wikipedia.

**Caption Consolidation:** Accumulating information across viewpoints to form a complete picture of 3D objects is challenging, but crucial. We find prompting of GPT4 *OpenAI* (2023a) to summarize the $M$ captions results in good parsing of the details across captions. By applying GPT4 as the final summary step, it can both include significant details and remove unlikely ones.

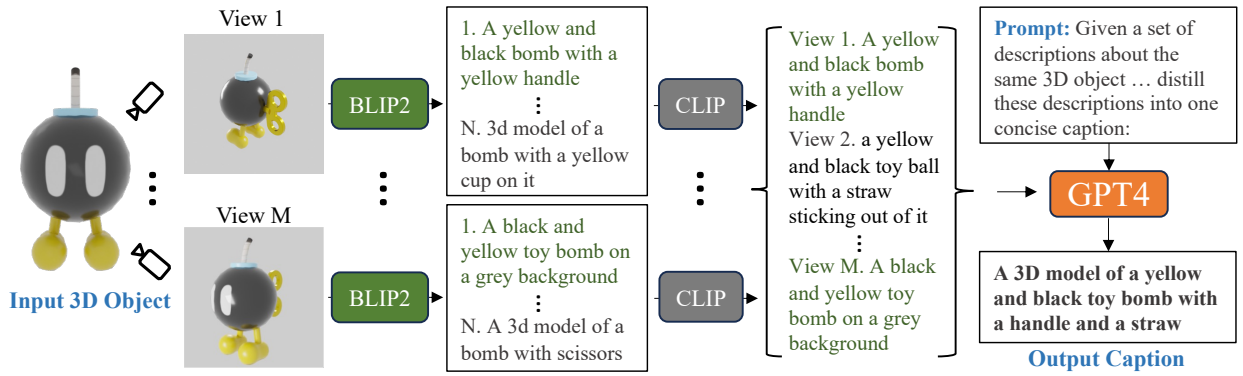



Figure 3.2: **Overview of Cap3D**. Left to Right: (1) Render 3D objects from $M = 8$ camera angles to capture object details (2) Generate $N = 5$ image captions per rendered image using BLIP2; (3) Select one caption for each image based on its similarity to the image encoding using CLIP; (4) Use GPT4 to consolidate all selected captions into a final, summary of the object.

For example, the final caption in Figure 3.2 filters the incorrect information, from view 2, “toy ball", while keeping key details, including "handle" and "straw". The alternative order of GPT4 followed by CLIP would result in (1) GPT4 having to make sense of more incorrect input details and (2) CLIP simply selecting between aggregate captions instead of being able to error-correct small mistakes. The effectiveness of introducing GPT4 is verified in ablations (Table 3.3).

### 3.3.2 Ethical Filtering

Captions generated and images rendered by Cap3D enhance the identification and mitigation of legal and ethical issues associated with large-scale 3D object datasets, including identifiable information and NSFW content.

We manage two datasets: Objaverse and ABO. In Objaverse, our main responsibility involves dealing with artist-created assets. These can include identifiable elements such as human face scans and NSFW objects. Objaverse contains approximately 800k objects, which makes the manual verification of each asset impractical. The ABO dataset, on the other hand, is smaller and mostly consists of furniture. We manually ensure the ethical integrity of this dataset.

We exclude objects that lack sufficient camera information for rendering, leaving us with 785k objects. These objects encompass a mix of non-commercial and commercial licenses, such as CC

BY-NC-SA, CC BY-NC, CC BY, CC BY-SA, and CC0. We then applied ethical filtering exclusively to objects with commercial-friendly licenses (i.e., CC BY, CC BY-SA, and CC0), resulting in a count of 680k objects.

We next follow prior work *Desai et al.* (2021) and use a face detector *Deng et al.* (2020) and NSFW classifier *Szegedy et al.* (2016); *Laborde* on forward-facing object renders and filter detected objects with score $>= 0.9$. The face detector filters out 18.6k objects, and the NSFW classifier filters out 217 objects. Text is also carefully processed. Our final captions are the output of GPT4, which has been trained to filter out inappropriate or harmful content *OpenAI* (2023a). We run a standard blocklist *LDNOOBW* on its output, removing any object-caption pairs including blocked words. This filters out 226 objects. After all the filtering, we are left with 661k objects in the Objaverse dataset. We manually estimate detection precision and recall in Table 3.2. To summarize, our process detects over 19k objects, of which a nontrivial amount is accurately removed. We estimate roughly 1k face and less than 1k NSFW are missed, using a conservative standard (e.g. missed faces are typically sports cards), resulting in a 660k subset.

Table 3.2: **Ethical Filtering Analysis.** We manually detect faces and NSFW content to validate automated filtering. 16 of 17 missed face detections were sports cards.

| | **Detected** | **Precision** | | **Missed dets.** | |
|---|---|---|---|---|---|
| | **(Filtered)** | **5k** | **(%)** | **10k** | **680k** |
| Faces | 18.6k | 790 | 16% | 17 | ≈1k |
| NSFW | 217 | 102 | 47% | 12 | <1k |
| Language † | 226 | – | – | – | – |

*†: String match filtering is deterministic.*

## 3.4 Dataset

We collect captions in two distinct settings: Objaverse, a large and varied dataset of artist-created 3D assets; and ABO, a small dataset of real products, typically furniture.

### 3.4.1 Objaverse Captions

Objaverse *Deitke et al.* (2023a) features roughly 800k 3D object assets across 21k classes designed by over 100k artists. It is of significantly larger scale than prior work; the paper shows

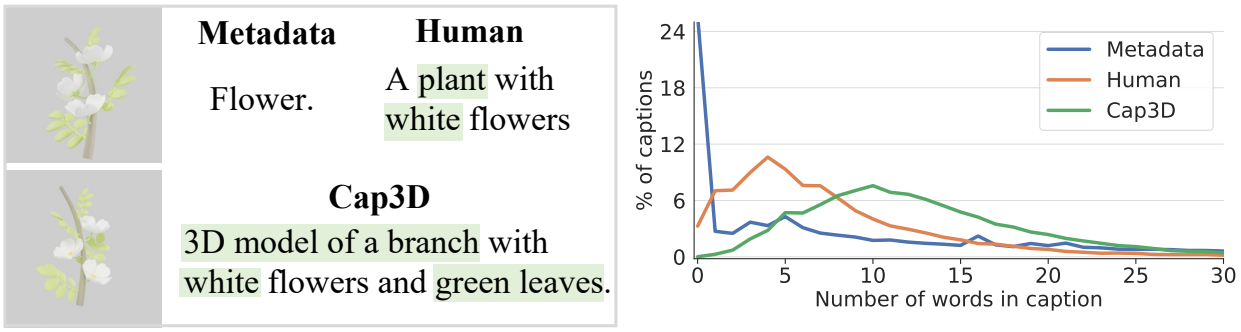



Figure 3.3: **Objaverse Caption Comparison.** Human captions and Internet metadata frequently contain limited detail. Cap3D captions typically have longer length and more detail. Vocabular size comparisons among Cap3D, BLIP2, and human captions are included in Appendix B.5.

this size enables more diversity by generative 3D models trained upon it. It is released under the ODC-By 1.0 license, permitting subsequent researchers to curate new data from it. Metadata is paired with many assets, however as seen in Figure 3.3 (right), metadata caption length is frequently short or empty. We collect two caption datasets on Objaverse. First, an automated set of one caption for each of 785k objects using Cap3D (a total of 785k captions). Second, a crowdsourced set of 41.4k captions spanning 39.7k objects for evaluating generated captions. Captions are collected using thehive.ai, a crowdsourced platform similar to AMT. Workers are given instructions with gold-standard sample captions, see the same 8 views as models during captioning, and are routinely monitored. Poor captioning performance results in a ban and deletion of the worker's captions. Crowdsourced captions are also filtered using the blocklist in § 3.3.2. Figure 3.3 (left) shows human captions provide more detail than metadata, but automated captions tend to be most descriptive.

### 3.4.2 ABO Geometry Captions

ABO *Collins et al.* (2022) is a collection of 3D models of Amazon products and is primarily furniture. ABO serves as an important contrast to Objaverse as it consists of a small number of classes varying primarily in geometry. Captioning, therefore, needs to focus more on structure as opposed to semantic category. To emphasize this focus, we consider the task of captioning the geometric structure of objects without color or texture (seen in the bottom right of Figure 3.1). Like

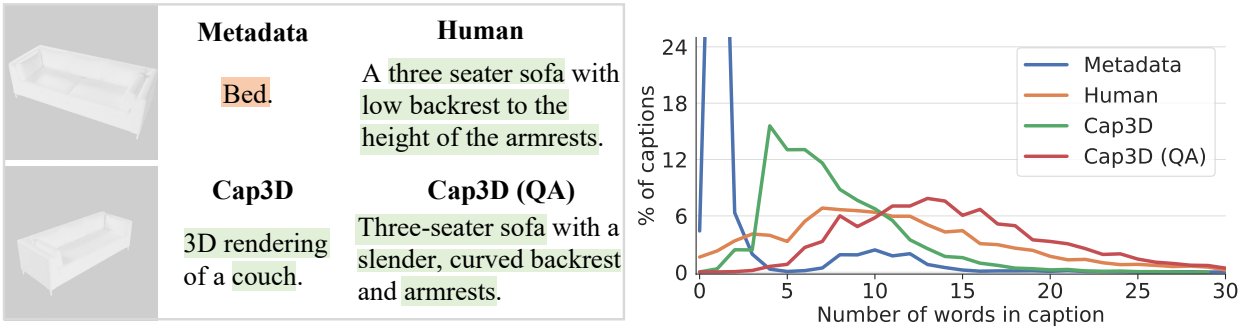



Figure 3.4: **ABO Automated Geometric Description**. **Left:** Human descriptions provide more detailed geometry than automated captions. With careful prompting, *Cap3D (QA)* can match human-level detail. **Right:** The high peak of Metadata is cropped, which otherwise obscures other curves.

Objaverse, ABO contains metadata that is typically quite short (Table 3.4), resulting in limited detail. We collect three sets of captions on the 6.4k ABO splits of *Luo et al.* (2023a): crowdsourced (a total of 17.2k captions), captions generated by Cap3D (a total of 6.4k captions), and captions generated by Cap3D (QA) which uses the two-stage prompt captioning (a total of 6.4k captions). Crowdsourced captions follow similar detail to Objaverse with the exception instructions and examples are focused on geometric structure. We compare alternatives in Figure 3.4. In contrast to Objaverse, human geometric descriptions on ABO are more detailed than captioning. With prompting (QA), the Cap3D pipeline can rival human descriptions.

## 3.5 Experiments

In this section, we first validate the quality of Cap3D captions against metadata and human-authored captions on both Objaverse and ABO. To verify Cap3D captions are helpful in practice, we next compare text-to-3D models finetuned on both human-authored captions and Cap3D (using the same >30k set as crowdsourced captions). Finally, we evaluate state-of-the-art text-to-3D models on our captions at scale to measure if finetuning on our captions can improve performance.

### 3.5.1 3D Captioning on Objaverse

**Dataset** We evaluate caption quality on three subsets of Objaverse: (1) a random set of 22k objects containing a human caption, (2) a random split of 5k objects containing a human caption, and (3) a random 5k split across the entire dataset.

**Baselines** In data splits (1) and (2), we compare the caption generated by Cap3D with human-authored annotations, *Human*, and existing Objaverse metadata, *Metadata*, described in § 3.4.1. Split (1) is used for A/B testing of *Cap3D* vs. *Human*, as shown in Table 3.1, at scale. Collecting A/B comparison is expensive, so we compute more extensive experiments on the smaller set (2) in Table 3.3.

In data split (3), we ablate the main components of Cap3D into *BLIP2* and *+GPT4*. *BLIP2* uses only the image captioning component of our method, taking a front-view rendering and producing a single output caption. *+GPT4* uses the same image captioning process of our method, producing 5 captions for each of 8 views. However, instead of using CLIP to filter 5 captions from each view, it directly summarizes all 40 captions into a final caption.

**Metrics** Our primary metric is human judgment A/B tests, where we ask workers to select between two captions on a scale of 1-5, where 3 is a tie. Workers are carefully monitored and each comparison has at least 10k observations across 5k objects.We report mean score, along with the percent each method is preferred (i.e. scores a 4 or 5). We use automated metrics CLIPScore *Hessel et al.* (2021); *Radford et al.* (2021b), the cosine similarity of CLIP encodings with input images; and ViLT Image and Text Retrieval, which ranks likely image-text pairs, from which one computes precision.

We emphasize CLIPScore is not our primary metric since our captioning model utilizes CLIP. BLIP2 utilizes ViT-L/14 and ViT-g/14, while our filtering uses ViT-B/32, so following previous work *Jain et al.* (2022) we compute CLIP score using a different model to reduce bias (ViT-B/16). However, we report it as it has shown a higher correlation with human judgments than other automated metrics *Hessel et al.* (2021). ViLT *Kim et al.* (2021b) is trained on different data and is a different architecture than CLIP, providing an orthogonal metric.

Table 3.3: **Objaverse Captions Evaluations**. *Cap3D* outperforms *human* and *Metadata*; *BLIP2*, *GPT4*, and *CLIP* are all important to performance. We report 95% confidence interval and use 5k objects.

| Method | User A/B Study vs. Cap3D | | | CLIP | ViLT Img Retr. | | ViLT Text Retr. | |
|---|---|---|---|---|---|---|---|---|
| | Score (1-5) | Win % | Lose % | Score | R@5 | R@10 | R@5 | R@10 |
| Metadata | 1.74±0.026 | 10.7 ± 0.7 | 83.8 ± 0.8 | 66.8 | 4.3 | 6.3 | 6.1 | 8.5 |
| Human | 2.86±0.026 | 37.0±1.0 | 46.1±1.0 | 72.5 | 21.2 | 29.0 | 18.5 | 24.9 |
| Cap3D | - | - | - | **88.4** | **35.7** | **46.3** | **34.7** | **44.2** |
| BLIP2 | 2.87± 0.019 | 41.0± 0.7 | 50.6± 0.7 | 83.1 | 24.7 | 32.3 | 21.9 | 29.3 |
| + GPT4 | 2.94± 0.015 | 35.2± 0.6 | 40.8± 0.6 | 86.3 | **31.9** | 39.9 | 30.2 | 38.4 |
| + CLIP (Cap3D) | - | - | - | **86.9** | 31.1 | **40.2** | **30.3** | **38.6** |

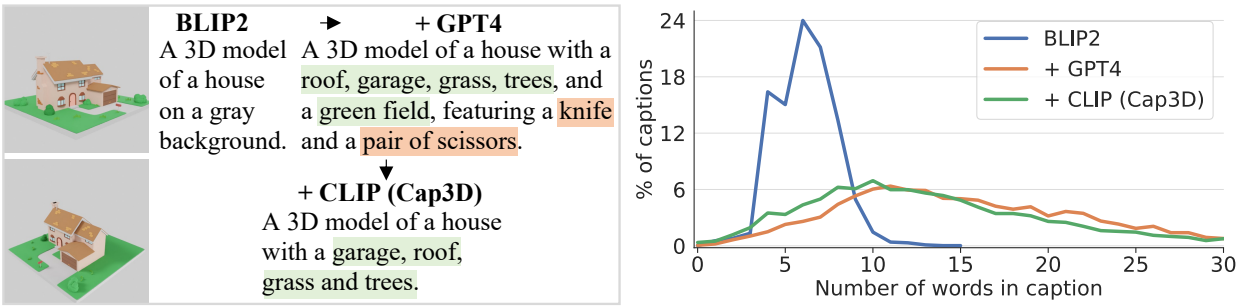



Figure 3.5: **Objaverse Caption Ablations**. GPT produces longer and more detailed captions than BLIP2; CLIP tends to prune incorrect details and reduces length slightly.

**Results** We report large scale A/B testing (1) against *Human* in Table 3.1, which shows Cap3D is better across metrics, with high confidence. The top three rows of Table 3.3 use the smaller human-captioned split (2), and demonstrate Cap3D's superior performance over Objaverse metadata and human-authored captions across A/B studies and automated metrics. The bottom three rows of Table 3.3, studied across a random split of the full dataset (3), reveal that while *BLIP2* is effective, incorporating multiple views with *+GPT4* enhances performance. As shown in Figure 3.5, GPT4 adds detail by consolidating view-specific information. Filtering using *+CLIP (Cap3D)* mitigates false details by purging subpar captions from GPT input. In addition to reducing errors, utilizing CLIP also reduces GPT input captions from 40 to 8, effectively decreasing token numbers and facilitating a cost reduction from $15.33 to $4.18.

### 3.5.2 Geometry 3D Captioning on ABO

**Dataset** We evaluate geometric captioning on a 6.4k object split from ABO *Collins et al.* (2022); *Luo et al.* (2023a), comparing Cap3D captions for each object against a maximum of two human-authored ones. To emphasize geometric focus, images used for model input and human assessment are texture-free and colorless.

**Baselines and Metrics** We use two automated variants from §3.3.1: *Cap3D* and *Cap3D (QA)*, which uses a two-stage prompt captioning to ask more about the input 3D geometry; and compare to crowdsourced human descriptions, *Human*, detailed in §3.4.1, and ABO metadata, *Meta*.

Our primary metric of comparison is similar human A/B testing to §3.5.1, since automated metrics such as CLIPScore do not accurately represent the distance between fine-grained captions and images as shown in *Luo et al.* (2023a).

**Results** In stark contrast to Objaverse, *Human* captions beat automated (*Cap3D*) in Table 3.4. Automated captions alone contain little geometric detail (e.g., Figure 3.4), making *Cap3D* unsuited for this setting. However, by using the two-stage prompt engineering, *Cap3D (QA)* is preferred to *Human*. Shown in Figure 3.4, *Cap3D (QA)* produces significant fine-grained geometric detail as well as longer captions in general. In contrast, *Metadata* is clearly the weakest baseline.

### 3.5.3 Large-Scale Text-to-3D Generation

**Dataset** We evaluate text-to-3D generation on three subsets of Objaverse: (1) a 30k split of objects containing human-authored captions, to measure if finetuning on Cap3D captions outperform human-authored ones; (2) a 350k split of Objaverse objects paired with Cap3D captions, for finetuning state-of-the-art text-to-3D methods – obtaining high-density point cloud and latent codes to finetune Point·E and Shap·E for all 785k objects is prohibitively expensive (20k GPU days); and (3) a 300 object split for optimization-based baselines, which typically take $>$30 mins per object to optimize. Pretrained and Finetuned models are evaluated on 8 views across a held-out test set of 2k objects.

**Methods** We consider several recent SOTA methods in three general categories: text-to-3D

Table 3.4: **ABO Fine-Grained Geometry Captions**. Cap3D (QA) performs best; crowd-sourced beats captioning alone.

| Method | A/B Score (1-5) | A/B Win % | A/B Lose % |
|---|---|---|---|
| Human v. Cap3D | 3.09±0.02 | 47.3±1% | 41.4±1% |
| Cap3D(QA) v. Human | 3.08±0.02 | 50.2±1% | 44.0±1% |
| Cap3D(QA) v. Cap3D | 3.27±0.02 | 56.0±1% | 37.4±1% |
| Cap3D(QA) v. Meta | 4.27±0.02 | 88.2±1% | 10.0±1% |

Table 3.5: **Text-to-3D: Human Captions**. Cap3D captions are better than human on the 30k set. Finetuning on Cap3D full set performs best.

| | Finetune Dataset | FID↓ | CLIP Score | CLIP R-Precision (2k) R@1 | R@5 | R@10 |
|---|---|---|---|---|---|---|
| Point·E | Pretrained | 36.1 | 72.4 | 6.0 | 16.2 | 22.4 |
| | 30k (Human) | 34.6 | 74.4 | 8.2 | 21.3 | 29.1 |
| | 30k (Cap3D) | 33.7 | 75.0 | 10.4 | 24.3 | 32.1 |
| | 350k (Cap3D) | **32.8** | **75.6** | **12.4** | **28.1** | **36.9** |
| Shap·E | Pretrained | 37.2 | **80.4** | **20.3** | **39.7** | **48.7** |
| | 30k (Human) | 36.0 | 79.6 | 18.6 | 36.3 | 45.3 |
| | 30k (Cap3D) | 37.2 | 79.4 | 19.1 | 37.5 | 46.1 |
| | 350k (Cap3D) | **35.5** | 79.1 | 20.0 | 38.8 | 47.3 |

diffusion, cascaded text-to-image then image-to-3D diffusion, and optimization-based. We use the direct text-to-3D variant of *Point·E Nichol et al.* (2022), as well as two variants of *Shap·E Nichol and Jun* (2023): *STF Gao et al.* (2022a) and *NeRF Mildenhall et al.* (2020). We use *Stable Diffusion* cascaded with *Point·E (Im-to-3D)*, adapting *ControlNet Zhang and Agrawala* (2023) and *LoRA Hu et al.* (2021) for Stable Diffusion finetuning. We use optimization-based baselines *DreamField Jain et al.* (2022), the publicly available implementation of *DreamFusion Poole et al.* (2022), Stable DreamFusion *Tang* (2022); and *3DFuse Seo et al.* (2023), using their implementation based on Karlo *Lee et al.* (2022); *Ramesh et al.* (2022).

**Metrics** We use standard metrics from prior work *Poole et al.* (2022); *Nichol et al.* (2022); *Nichol and Jun* (2023); *Jain et al.* (2022) to evaluate. Primarily, these are CLIP Score and CLIP R-Precision. CLIP R-Precision ranks a rendered image against all text pairs in the test set by CLIP cosine similarity, and computes precision upon true text-image correspondence. Since we have ground truth images, we calculate the FID *Heusel et al.* (2017) of 3D rendered images against ground truth images, as well as assess CLIP Score on these reference images. We also use ViLT Retrieval R-Precision, used in 3.5.1, which has the same evaluation procedure as CLIP R-Precision with a different model.

**Results** Table 3.5 lists the results of finetuning using human-authored and Cap3D captions. Point·E improves after finetuning upon human captions. However, performance is further improved using our captions on the same dataset; and improved most by training upon the full dataset. This

Table 3.6: **Text-to-3D on Objaverse**. Finetuning improves FID over pretrained performance across models. CLIP metrics of *Stable Diffusion* increase; CLIP metrics of *Point·E* increase significantly.

| | Pretrained | | | | | Finetuned on Cap3D | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | FID↓ | CLIP Score | CLIP R-Precision (2k) R@1 | R@5 | R@10 | FID↓ | CLIP Score | CLIP R-Precision (2k) R@1 | R@5 | R@10 |
| Ground Truth Images | - | 81.6 | 32.7 | 55.1 | 64.3 | - | 81.6 | 32.7 | 55.1 | 64.3 |
| Point·E (Text-to-3D) *Nichol et al.* (2022) | 36.1 | 72.4 | 6.0 | 16.2 | 22.4 | **32.8** | **75.6** | **12.4** | **28.1** | **36.9** |
| S. Diff. *Rombach et al.* (2022) (CNet) *Zhang and Agrawala* (2023)+ *Nichol et al.* (2022)(Im-to-3D) | 54.7 | 73.6 | 11.0 | 23.4 | 30.0 | **53.3** | **74.6** | **12.4** | **26.2** | **33.8** |
| S. Diff. *Rombach et al.* (2022) (LoRA) *Hu et al.* (2021)+ *Nichol et al.* (2022)(Im-to-3D) | 54.7 | 73.6 | 11.0 | 23.4 | 30.0 | **53.7** | **74.4** | **11.6** | **24.6** | **31.4** |
| Shap·E *Nichol and Jun* (2023) (STF) *Gao et al.* (2022a) | 37.2 | **80.4** | **20.3** | **39.7** | **48.7** | **35.5** | 79.1 | 20.0 | 38.8 | 47.3 |
| Shap·E *Nichol and Jun* (2023) (NeRF) *Mildenhall et al.* (2020) | 48.7 | **79.4** | **19.0** | **37.7** | **46.8** | **48.2** | 78.1 | 18.3 | 35.1 | 43.5 |

result strongly defends Cap3D captioning at scale. Shap·E does not improve on CLIP metrics after finetuning in any dataset, but performs the least bad on the full dataset using our captions; and FID improves most.

Table 3.6 presents results from several state-of-the-art pretrained and finetuned models using Cap3D-generated captions. The models finetuned on our captions generally outperform pretrained models under the FID metric. For CLIP-related metrics, the finetuned models of *Point·E (Text-to-3D)* and *StableDiffusion + Point·E (Im-to-3D)* also beat their pretrained counterparts. Point·E and Stable Diffusion have been trained on massive datasets, so improvement from finetuning is strong evidence Cap3D captions are effective. The observed downturns in *Shap·E* could be attributed to at least two factors. First, our replication of their privately-available train code is unstable, often resulting in NaN loss during finetuning. We restart from earlier checkpoints upon crashing, but the result alone is concerning. Second, we exclusively finetune the diffusion model in Shap·E's two-stage approach.

Qualitative results in Figure 3.6 validate quantitative findings. *Point·E* and *Stable Diffusion* baselines show large improvements from finetuning, while *Shap·E* can better fit the Objaverse data distribution (corresponding to improved FID).

Optimization baselines, shown in Table 3.7, perform very well upon CLIP-based metrics, consistent with prior work *Nichol and Jun* (2023). In fact, *DreamField* outperforms ground truth images in CLIP metrics. This demonstrates *DreamField* overfits to the CLIP metric, which is the standard protocol for text-to-3D

Table 3.7: **Text-to-3D: Optimization Baselines**. Overfitting via CLIP leads to higher CLIP-based scores than ground truth; ViLT score is more fair.

| | FID↓ | CLIP Score | CLIP R@1 | CLIP R@5 | ViLT R@1 | ViLT R@5 |
|---|---|---|---|---|---|---|
| True Images | - | 83.2 | 53.2 | 77.8 | 41.3 | 69.0 |
| D. Field *Jain et al.* (2022) | 106.1 | **83.7** | **61.8** | **83.6** | **32.3** | **56.0** |
| D. Fusion *Poole et al.* (2022) | 127.8 | 72.4 | 28.4 | 46.1 | 23.7 | 45.3 |
| 3DFuse *Seo et al.* (2023) | **91.1** | 77.0 | 38.6 | 58.5 | 26.3 | 53.0 |

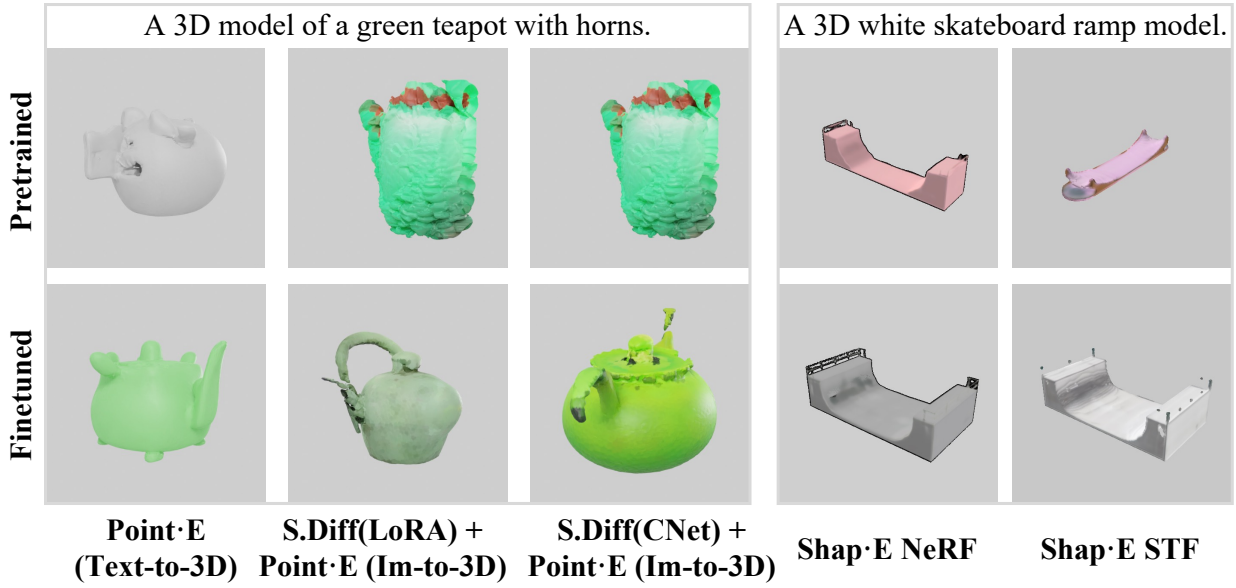



Figure 3.6: **Text-to-3D results.** Finetuning on Cap3D captions can significantly improve results. Additional examples are available in Appendix B.3.

evaluation. We propose to also consider ViLT precision (see §3.5.1). This helps mitigate the bias of CLIP, though *DreamField* performance on this metric is still strong.

## 3.6 Limitations and Future Works

As described in §3.3, Cap3D consists of four steps: (1) 3D objects rendering; (2) captioning via BLIP2; (3) filtering captions via CLIP; (4) consolidate multiview information via GPT4. To effectively capture comprehensive information through 2D renderings, cameras are strategically placed above or below objects. However, this occasionally results in unconventional 2D views, making BLIP2 susceptible to errors that CLIP fails to rectify. This, in turn, hampers GPT4's ability to merge variegated information across views, culminating in vague and verbose descriptions, as illustrated in Figure 3.7. The system also falters with certain complex indoor 3D scans, as depicted in Figure 3.8, thus requiring more robust image-captioning models *OpenAI* (2023b) and potentially benefiting from additional view incorporations beyond the current eight.

Our method's provision of extensive 3D-Caption pairs for Objaverse *Deitke et al.* (2023a)

could foster the advancement of 3D-LLM models (*Xu et al.*, 2023a; *Hong et al.*, 2023), facilitating 3D-caption centric tasks like captioning, dialog, and language-based navigation. Additionally, the geometric descriptions generated for ABO *Collins et al.* (2022) enable compositional structure analysis of fine-grained 3D objects *Luo et al.* (2020); *Mo et al.* (2019b). Our developed method assists in scaling up of 3D-text pairs for expansive 3D datasets *Deitke et al.* (2023b).

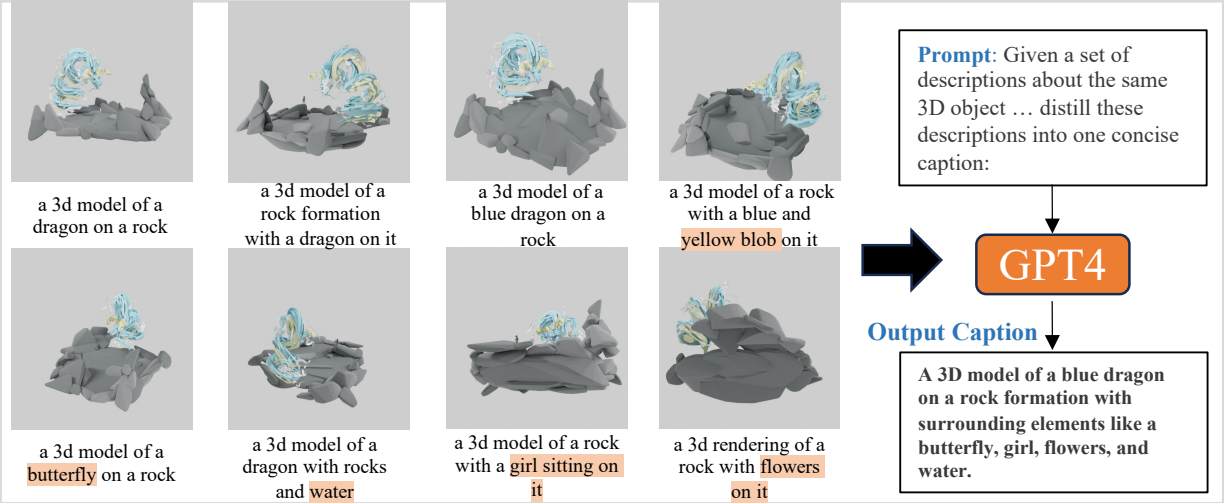



Figure 3.7: An failed case. The caption under each rendered image are generated by BLIP2 + filtered by CLIP. The inaccurate content are highlighted with colors. GPT4 + CLIP cannot fix the error generated by BLIP2 and result in a fuzzy description.

## 3.7 Conclusion

In this work, we collect (1) 3D object captions at scale, creating the largest publicly available high-quality 3D-text by an order of magnitude. To do so we propose Cap3D, an automated pipeline leveraging several models pretrained on large datasets, and show design choices are important to performance. In addition, we collect (2) a dataset of geometric captions upon fine-grained 3D objects. This helps analyze shortcomings of automated captioning and study the potential of question answering, while yielding geometric descriptions for 3D assets of real objects paired with real images. These datasets serve as benchmarks for text-to-3D tasks (1) at scale and (2) in geometric detail.

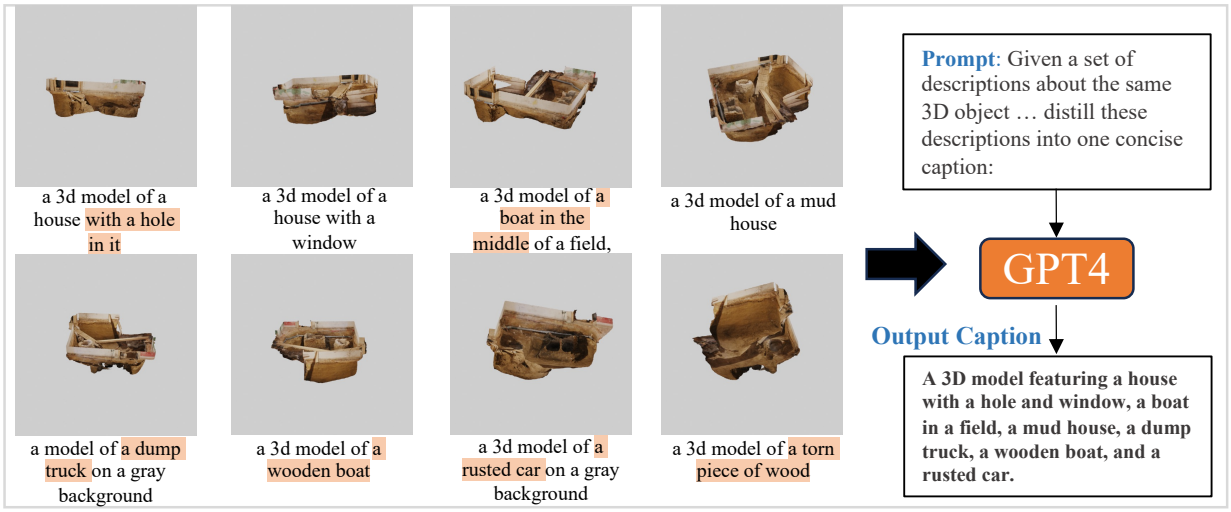



Figure 3.8: An failed case. The caption under each rendered image are generated by BLIP2 + filtered by CLIP. The inaccurate content are highlighted with colors. The various views contain inaccurate information. The associated details, roughly described, fail to accurately depict the indoor scene.

# CHAPTER 4

# View Selection for 3D Captioning via Diffusion Ranking

*Cap3D demonstrated that multi-view rendering can bridge 2D captioning models to 3D assets. However, not all views are equally informative: some camera positions yield "accidental" projections that confuse captioners and introduce hallucinations. This chapter refines the multi-view pipeline by introducing* ***structure-aware view selection****. Using signals from pretrained diffusion models, we rank views by their alignment with the underlying 3D content, prioritizing the most informative perspectives and suppressing misleading ones.*

## 4.1 Introduction

Recent advancements in generative models have shown remarkable performance in both image *Saharia et al.* (2022); *Betker et al.* (2023) and video *Brooks et al.* (2024) domains, driven by the availability of extensive captioned datasets. Despite these successes, extending generative modeling to 3D domains has been challenging due to the scarcity of high-quality 3D-text pairs. This gap has been partially bridged by Cap3D *Luo et al.* (2023b), which generates captions for 3D objects by rendering them into 2D images and employing image-based captioning models, further refined by Large Language Models (LLMs) to synthesize captions. Cap3D has contributed 660k captions

This chapter is based on work published at ECCV 2024 (*Luo et al.*, 2024).

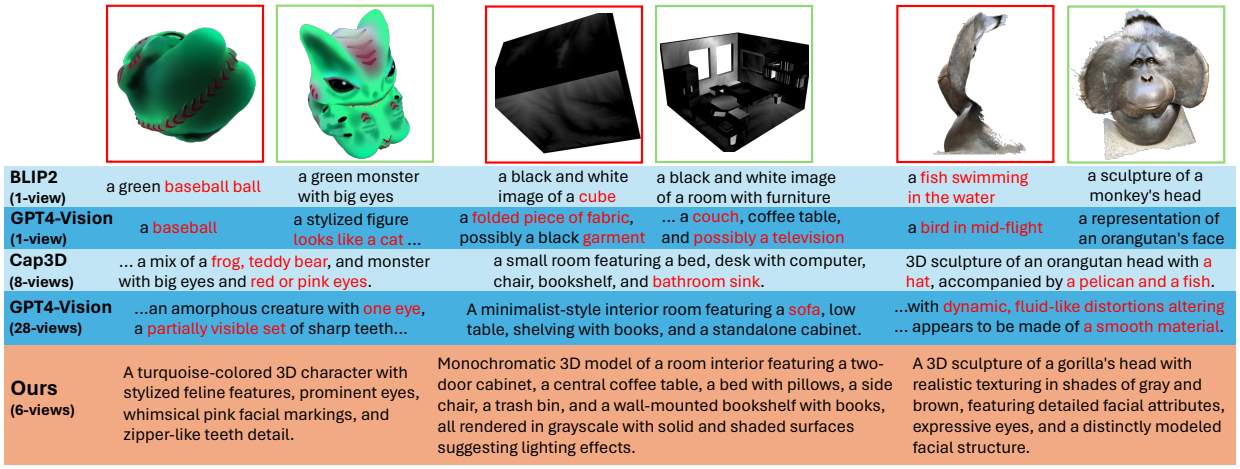



Figure 4.1: DiffuRank enhances caption accuracy and reduces hallucinations by prioritizing key rendered views (green box), contrasting the accidental views *Freeman* (1994) (red box) that cause errors. Surprisingly, using fewer views (6 vs. 28) not only saves computational resources but also may yield more accurate and detailed outcomes (the middle example) by countering the uncertainty caused by excessive views.

for the Objaverse dataset *Deitke et al.* (2023a), facilitating developments in Text-to-3D *Yariv et al.* (2023); *Li et al.* (2023f), Image-to-3D *Xu et al.* (2023b); *Zhao et al.* (2023a), robot simulator *Wang et al.* (2023b) and learning *Qi et al.* (2024), and the pre-training of 3D LLMs *Xu et al.* (2023a); *Zhou et al.* (2023); *Panagopoulou et al.* (2023).

Despite the utility of Cap3D, our analysis reveals that a significant portion of Cap3D captions contain inaccurate or hallucinated details, potentially compromising model training *Tang et al.* (2023b). Upon inspection, we identified one of the root causes as *accidental views* – special camera positions where coincidental feature alignments make the projection uninformative or misleading for recognition. Examples include the first “baseball" image, which actually depicts a green creature, and the third “cube" image, which is in fact an indoor design, both shown in Figure 4.1. Because Cap3D inherits Objaverse’s default object orientation, its cameras sit on a fixed horizontal ring, a heuristic that inevitably samples such accidental views. These accidental renderings are hard even for humans, and current captioners *Li et al.* (2023d) struggle to describe them faithfully. Consequently, when these challenging views are included, even advanced captioning models like GPT4-Vision *Achiam et al.* (2023b) may generate erroneous information, as illustrated in Figure 4.1.

To address this, we introduce DiffuRank, an approach for ranking rendered views with pre-

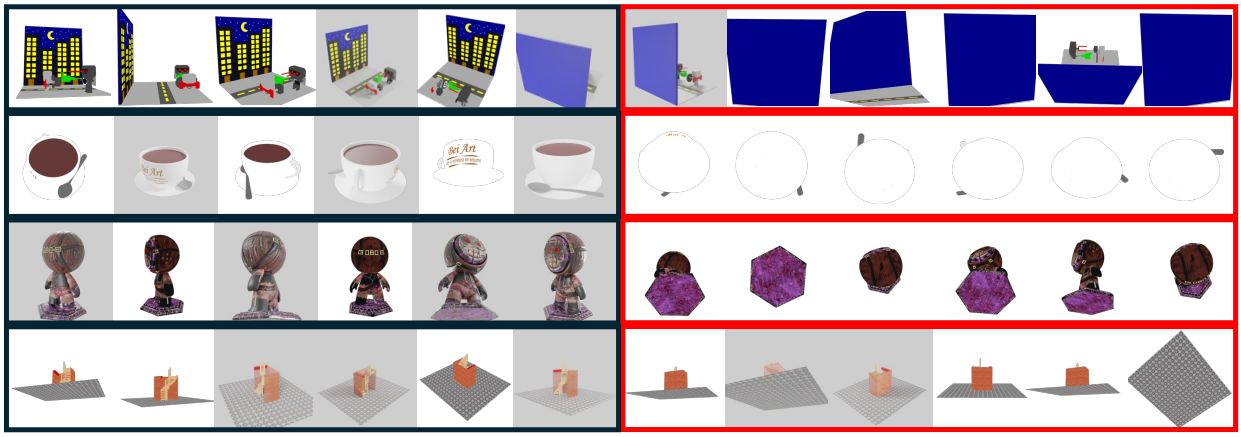


Figure 4.2: The left row features the top-6 views as ranked by DiffuRank, while the right row displays the bottom-6. Comparative analysis shows that the top-6 views generally uncover more characteristics of the object compared to the bottom-6. This finding underscores DiffuRank's capability to identify views that more accurately represent the features of the 3D object. More randomly sampled results are included in Appendix B.5.

trained diffusion models. By leveraging a pre-trained text-to-3D diffusion model *Jun and Nichol* (2023), DiffuRank evaluates the alignment between the captions of each view and the corresponding 3D object's information. The underlying premise is that captions generated from rendered views that closely match the object's 3D information will exhibit a higher alignment, indicating these views are more representative of the object. Consequently, DiffuRank promotes the preferable views (Figure 4.2) for captioning as those that better reflect the true essence of the 3D objects, leading to more accurate and truthful captions.

Specifically, we first employ image-based captioning models to caption all candidate rendered views. We then utilize multiple iterations of a text-to-3D diffusion model objective to estimate an average alignment score, using each caption as a conditioning input for the same underlying 3D object feature, Gaussian noise, and timestamps. The intuition here is that captions better aligned with the 3D object feature provide more effective guidance during the diffusion process, resulting in higher alignment scores. This score effectively quantifies how well each caption aligns with the corresponding 3D object feature. Since the captions are generated from rendered views via captioning models trained through maximum likelihood estimation, the resulting alignment scores inherently reflect the alignment between the rendered views and the original 3D object. Higher

scores indicate stronger alignment, implying fewer possibilities of being accidental views.

Following this scoring procedure, we rank the views based on their alignment scores and select the top-N rendered views for further caption generation using GPT4-Vision. Human evaluations demonstrate that captions produced using DiffuRank combined with GPT4-Vision exhibit significantly higher quality and fewer inaccuracies compared to captions generated by Cap3D. Additionally, our method consistently yields richer, more detailed captions and fewer hallucinations when employing just 6 strategically selected rendered views, as opposed to using GPT4-Vision directly on all 28 views or those selected by default object orientations.

Further, we extend DiffuRank to the 2D domain, demonstrating its effectiveness in the challenging Visual Question Answering task *Tong et al.* (2024a) when combined with text-to-2D diffusion models *Rombach et al.* (2022), and surpassing the zero-shot performance of CLIP *Radford et al.* (2021b).

Our contributions are as follows:

- We propose DiffuRank to address the issue of accidental views in 3D object renderings. In principle, DiffuRank estimates the alignment score between different modalities. Therefore, we extend DiffuRank to the 2D domain and demonstrate that, with the assistance of a pre-trained text-to-2D diffusion model *Rombach et al.* (2022), DiffuRank outperforms CLIP on the VQA task *Tong et al.* (2024a).

- We identify and alleviate the systematic hallucinations in Cap3D captions, revising approximately 200k entries with the help of DiffuRank and GPT4-Vision. The corrected captions consistently improve the finetuned performance of text-to-3D models (Point·E, Shap·E); note that Shap·E models fine-tuned with Cap3D captions show decreased performance.

- We extend the Cap3D caption dataset *Luo et al.* (2023b) from 660k to 1.5M across the entire Objaverse *Deitke et al.* (2023a) and a portion of the Objavere-XL high-quality subset *Deitke et al.* (2023b). Each caption is complemented with point clouds containing 16,384 colorful points and 20 rendered images, including camera, depth, and MatAlpha details. All data is re-

leased under the ODC-By 1.0 license and is available at https://huggingface.co/datasets/tiange/Cap3D.

## 4.2 Related Work

### 4.2.1 3D-Text

Recent advancements introduced by Objaverse have significantly enriched the field of 3D object research. By integrating a comprehensive set of 3D objects with descriptive captions from Cap3D, a wide array of 3D applications has been enabled. These include Text-to-3D methods *Yariv et al.* (2023); *Li et al.* (2023f); *He et al.* (2023); *Li et al.* (2023g); *Mercier et al.* (2024), Image-to-3D conversion techniques *Xu et al.* (2023b); *Zhao et al.* (2023a), enhancements in robot learning *Wang et al.* (2023b); *Qi et al.* (2024), the pre-training of 3D language models *Xu et al.* (2023a); *Zhou et al.* (2023); *Qi et al.* (2023); *Liu et al.* (2024a); *Chen et al.* (2023d), and the development of language models capable of processing diverse modalities *Han et al.* (2023); *Panagopoulou et al.* (2023); *Chen et al.* (2024a).

Despite these advancements, we identified issues with hallucination contents in the captions provided by Cap3D. This discovery aligns with findings from concurrent research *Tang et al.* (2023b); *Liu et al.* (2023j); *Kabra et al.* (2023), pinpointing inaccuracies in Cap3D captions. Our investigation reveals that the root cause of these inaccuracies is attributed to atypical rendered views, which lead to failures in captioning models. These failures are exacerbated as text summarization models (GPT4) are unable to rectify these errors. To address this challenge, we introduce DiffuRank that selects rendered views capturing the essential characteristics of 3D objects. Furthermore, we utilize the recent advancements in vision-language models, specifically GPT4-Vision, to provide holistic captions for 3D objects. We release our dataset under ODC-By 1.0 license to enable research and commercial usage, and hope facilitate related 3D-Text research *Jain et al.* (2021); *Poole et al.* (2022); *Lin et al.* (2022); *Sanghi et al.* (2022); *Zhu and Zhuang* (2023); *Wang et al.* (2023a); *Chen et al.* (2023c); *Lorraine et al.* (2023); *Li et al.* (2023f); *Yi et al.* (2023); *Li et al.* (2023g); *Luo et al.* (2023a); *Ding et al.* (2023); *Chen et al.* (2023e); *Michel et al.* (2022); *Wei et al.* (2023); *Chen et al.*

(2023a); *Nichol et al.* (2022); *Liu et al.* (2023g,f); *Melas-Kyriazi et al.* (2023); *Tang et al.* (2023a); *Shi et al.* (2023); *Xu et al.* (2023b); *Chen et al.* (2023f); *Shi et al.* (2023).

### 4.2.2 Diffusion Model

Our proposed DiffuRank leverages denoising diffusion objective *Sohl-Dickstein et al.* (2015); *Song and Ermon* (2019); *Ho et al.* (2020) to model the alignment between the input and output modalities. By using pre-trained text-to-3D *Nichol et al.* (2022); *Jun and Nichol* (2023) and text-to-2D *Saharia et al.* (2022); *Betker et al.* (2023); *Peebles and Xie* (2023) diffusion models, we can model the alignment between given 3D object/image for a set of possible captions (text descriptions) as detailed in Section 4.3.2. In our listed algorithm 1, we adopt the objects $L_{3D} = E_{x_0 \sim q(x_0), \varepsilon \sim \mathcal{N}(0,\mathbf{I}), t \sim U[1,T]} \left\| x_\theta \left( x_t, t \right) - x_0 \right\|_2^2$ as used in Shap·E *Jun and Nichol* (2023), where $x_0$ is data sampled from data distribution $q(x_0)$, $\varepsilon$ is Gaussian noise, and t is timestamp. We also adopt the alternative but equivalent objective, $L_{2\mathrm{D}} = E_{x_0 \sim q(x_0), \varepsilon \sim \mathcal{N}(0,\mathbf{I}), t \sim U[1,T]} \left\| \varepsilon - \varepsilon_\theta \left( x_t, t \right) \right\|_2^2$, when we adopt the text-to-2D model, stable-diffusion, in Section 4.5.3.

DiffuRank is related to score sampling distillation proposed in *Poole et al.* (2022), while we do not compute gradients but sampling loss to accumulate scores estimation for ranking. Our findings also relate to works which leverage pre-trained diffusion models for downstream tasks, such as image classification *Mukhopadhyay et al.* (2023); *Li et al.* (2023a), semantic segmentation *Zhao et al.* (2023b), visual grounding *Liu et al.* (2023h), depth prediction *Saxena et al.* (2023); *Zhang et al.* (2024a), and other low-level computer vision tasks *Du et al.* (2023).

When applying our method to the 2D domain, we discovered that our algorithm aligns closely with the insights of the approach presented in *Li et al.* (2023a). Consequently, our method can be considered an expansion of the findings from *Li et al.* (2023a), extending its applicability from 2D classification to broader domains and tasks, including the use of a pre-trained text-to-3D diffusion model and a 2D-image-based captioning model to estimate the alignment between 3D objects and their 2D rendered views, as well as the application of a pre-trained text-to-2D diffusion model to solve Visual Question Answering tasks.

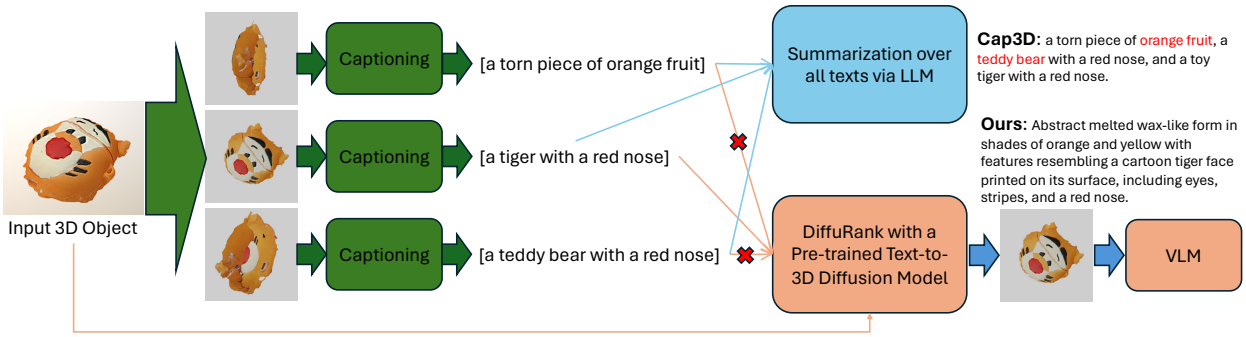



Figure 4.3: Methods overview. Both Cap3D and our method render input 3D objects into multiple views for caption generation (green steps). However, while Cap3D consolidates these captions into a final description (blue steps), our method employs a pre-trained text-to-3D diffusion model to identify views that better match the input object's characteristics. These selected views are then processed by a Vision-Language Model (VLM) for captioning (orange steps).

## 4.3 Method

In this section, we analyze the issues with atypical rendered views leading to hallucinations in Cap3D captions, motivating our proposed DiffuRank, a approach for selecting informative rendered views with 3D priors learned from a diffusion model. We then detail DiffuRank's formulation and describe our novel 3D captioning framework that integrates GPT4-Vision.

### 4.3.1 Issues in Cap3D

Firstly, we revisit the Cap3D pipeline, which unfolds across four stages. Initially, it renders a set of 2D views for each 3D object. Subsequently, image captioning is applied to generate preliminary descriptions (5 captions for each image). Then, the CLIP model is utilized in the third stage to select the best-aligned caption for each image, filtering out inaccuracies. The process culminates with an LLM synthesizing captions from various perspectives into a comprehensive caption.

However, the captioning of rendered views (the combined second and third stages) for given 3D objects can falter with atypical views, producing captions that diverge significantly from the actual 3D object. In the worst-case scenarios, each rendering view might correspond to an incorrect object, leading to compounded errors when these captions are summarized by GPT4. One example is shown in Figure 4.3. Since GPT4 operates solely on text, it cannot correct these inaccuracies,

resulting in captions riddled with hallucinated details.

Due to the versatility of 3D object geometries, determining which rendered views best reflect a 3D object's characteristics is non-trivial. While measuring the geometric properties of 3D objects and computing their principal directions is feasible, positioning the camera orthogonally, as shown in the bottom-left example of Figure 4.2, is often suboptimal. Hence, we propose DiffuRank, which learns 3D priors from data to filter preferable rendered views by leveraging a pre-trained text-to-3D model. Our experiments demonstrate that DiffuRank efficiently enhances caption quality and reduces hallucinations with fewer renderings compared to using all available views.

### 4.3.2 DiffuRank Formulation

DiffuRank leverages a pre-trained text-to-3D diffusion model $D_{text-to-3D}$ to rank rendered views based on their alignment with both captions and the corresponding 3D information.

For a given 3D object $\mathscr{O}$, assuming a set of candidate captions $c_i$ and the pre-trained model $D_{text-to-3D}$, the training objective of this pre-trained diffusion model is predicting a 3D object $\mathscr{O}$ based on a text description $c$, i.e., modeling the score function $\nabla_{\mathscr{O},c} p(\mathscr{O}|c)$ of the data distribution $p(\mathscr{O}_i|c)$. Specially, the diffusion model aims to minimize

$$\mathscr{L}_c = \|D_{text-to-3D}(\mathscr{O}_t|c) - \mathscr{O}_0\|$$

based on a text description $c$, where the noised input $\mathscr{O}_t = \sqrt{\bar{\alpha}_t}\mathscr{O} + \sqrt{1-\bar{\alpha}_t}\varepsilon$, for timestamp $t$, and randomly sampled Gaussian noises $\varepsilon \sim \mathscr{N}(0, I)$, with $\bar{\alpha}$ being a hyper-parameters defined by the noise schedule *Ho et al.* (2020). Our tuition here is simple: a caption closely aligned with the given 3D object in terms of characteristics (e.g. structure, colors, textures, etc), should aid the diffusion model in making accurate predictions starting from the same noised input $\mathscr{O}_i^t$, resulting in a lower score matching loss. By sampling multiple sets of $t_j, \varepsilon_j$ for the same set of captions $c_i$, we can measure the alignment $Cor(\mathscr{O}, c_i)$ between the 3D object and captions via the average loss.

Initially, we generate candidate captions for $\mathscr{O}$ by rendering it into multiple views $I_i$ and

**Algorithm 1** DiffuRank for modeling the alignments between 3D object and its rendered views

**Require:** Given 3D object $\mathscr{O}$, pre-trained text-to-3D model $D_{\text{text-to-3D}}$, captioning model $D_{\text{cap}}$

▷ 1. Rendered views $\{I_i\}_{i=1,\cdots,M}$ for $\mathscr{O}$ with a rendering program (e.g., Blender).

▷ 2. Generate candidate captions for $\mathscr{O}$.

**for** each view $I_i$ of $\mathscr{O}$ **do**

    Generate captions $\{c_i^j\}_{j=1,\cdots,N}$ with captioning model $D_{\text{cap}}$.

▷ 3. Compute average alignment scores.

**for** $k \leftarrow 1$ to num_samples **do**

    Sample timestamp $t_k \sim \text{Uniform}(0,1)$.

    Sample noise $\varepsilon_k \sim \mathscr{N}(0,I)$.

**for** each rendering view $I_i$ **do**

    **for** $k \leftarrow 1$ to num_samples **do**

        Compute noised input $\mathscr{O}_{t_k} = \sqrt{\bar{\alpha}_{t_k}}\mathscr{O}_0 + \sqrt{1-\bar{\alpha}_{t_k}}\,\varepsilon_k$.

        **for** $j \leftarrow 1$ to $N$ **do**

            Compute loss $\mathscr{L}_{c_i^j,k} = \|D_{\text{text-to-3D}}(\mathscr{O}_{t_k} \mid c_i^j) - \mathscr{O}_0\|$.

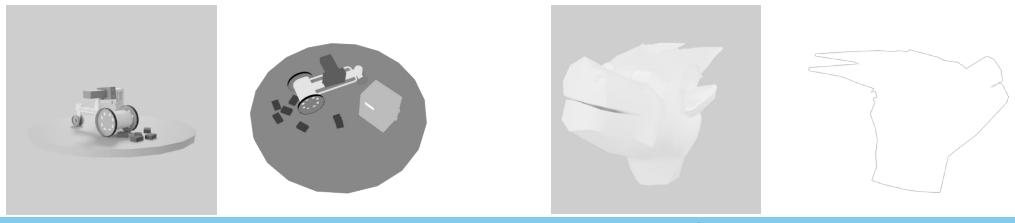
    Compute average loss for all captions of $I_i$,
    $Cor(I_i,\mathscr{O}) = -\mathbb{E}_{j,k}\mathscr{L}_{c_i^j,k}$.

**return** Top-P($\{Cor(I_i,\mathscr{O})\}_{i=1,\cdots,M}$)

generating captions $c_i^j$ with a captioning model $Dcap$. This captioning procedure aims to maximize the joint likelihood of the model distribution $p(c_i^j, I_i)$ over the image $I_i$ and generated captions $c_i^j$. Thus, we estimate the alignment between the 3D object and all captions of the same rendering $Cor(\mathscr{O}, \mathbb{E}_j c_i^j)$, which is proportional to $Cor(\mathscr{O}, \mathbb{E}_j p(c_i^j, I_i)) \propto Cor(\mathscr{O}, I_i)$. Then, we write down the whole pipeline in Algorithm 1.

Specifically, we adopted shap-E as the text-to-3D diffusion model in our paper, and the above $\mathscr{O}_i$ should be $E_{encoder}(\mathscr{O}_i)$, where $E_{encoder}$ is the encoder (transmitter in *Jun and Nichol* (2023)) to extract feature embeddings from given 3D object.

Furthermore, DiffuRank's application is not confined to 3D captioning; because it is a general framework for measuring the alignment between two modalities received and output by a diffusion model. It can be seamlessly extended to other domains, such as 2D images. In section 4.5.3, we show an example where we apply DiffuRank to perform 2D VQA and beat CLIP model *Radford et al.* (2021b).



| | | |
|---|---|---|
| **Cap3D** **(8-views)** | a 3d model of a small wheeled machine with a box on it. | a white 3d model of a head, resembling a dragon, wolf, and bull. |
| **Ours** **(6-views)** | Monochromatic 3D model of an industrial cannon-type machine with various components including a cylindrical barrel, a control box with buttons, and assorted geometric shapes that appear to be ammunition or parts scattered nearby, all positioned on a circular base. | abstract white geometric model resembling a stylized dragon head with pointed ears and faceted surfaces. |

Figure 4.4: We utilized both grey background + ray-tracing render engine (left images) and transparent background + real-time render engine (right images) for rendering, discovering that the effectiveness of each varies. We noticed DiffuRank can select the views with the appropriate rendering that highlight object features.

### 4.3.3 New 3D Captioning Framework

With the proposed DiffuRank, we establish a new 3D captioning pipeline, as shown in Figure 4.3. For given 3D object, we render it into 28 images, which are then captioned into 5 descriptions using an image-based captioning model. Following captioning, DiffuRank ranks the rendered views using a pre-trained text-to-3D model. This ranking enables the selection of the Top-6 rendered views for processing by a vision-language model, resulting in holistic captions that describe structure, form, color, and texture, with enhanced accuracy and detail.

To elaborate, our methodology integrates two distinct rendering strategies, as illustrated in Figure 4.4. The first strategy, derived from Cap3D *Luo et al.* (2023b), renders objects into 8 views against a uniform grey background, arranged horizontally around the object's default orientation, with Blender ray-tracing render engine 'CYCLES'. Concurrently, we apply a second technique from Shap·E *Jun and Nichol* (2023), where 20 views are generated through randomized sampling after object normalization, set against a transparent background, with Blender real-time engine 'EEVEE'. These 20 views, created following the Shap·E methodology, are instrumental in forming Shap·E latent codes, i.e. $E_{encoder}(\mathcal{O}_i)$ in Section 4.3.2. Altogether, this approach results in 28 distinct views for each object. Additionally, as grey and transparent backgrounds may accentuate or obscure details variably across objects, we observed that DiffuRank adeptly selects the views with the proper

background that most effectively highlight object features, without manual intervention. Some examples are included in Appendix C.1.

Following this, the captioning model, BLIP2 *Li et al.* (2023d), is employed to generate five captions for each view. These captions, alongside the pre-trained text-to-3D diffusion model, Shap·E *Jun and Nichol* (2023), and the previously derived 3D latent code $E_{encoder}(\mathcal{O}_i)$, undergo analysis in the DiffuRank process, as detailed in Algorithm 1. Subsequent to DiffuRank, the six views that demonstrate the highest alignment scores are chosen to input into GPT4-Vision for caption generation.

## 4.4 Dataset

This section details our process for correcting the Cap3D captions, expanding the dataset with high-quality 3D objects from Objaverse-XL, and ethical filtering. More detailed hyper-parameters and comparisons are included in Appendix C.1.

### 4.4.1 Correction of Cap3D Captions

As Cap3D contains a lot of good quality captions, our first objective is to identify erroneous Cap3D captions, which might contain incorrect information or hallucinations. We tried three strategies as outlines the below.

**Image-Text Alignment Method:** We discovered that utilizing the maximum and average CLIP scores effectively filters out inaccurate captions. Most of erroneous captions, like those depicted in Figure 4.1, described improbable combinations of objects (e.g., “a mix of a frog, teddy bear, and monster" or “an orangutan accompanied by a pelican and a fish") in scenarios where only one entity was present in the given 3D object. Such discrepancies arise when different views of the same 3D object receive varied entity captions from BLIP2, which GPT4 then erroneously combines, shown in Figure 4.3. To detect this kind of case, we computed both the average and maximum CLIP scores between the final caption and all eight rendered views used in Cap3D. A validation set of $\sim 7k$ objects with inaccurate captions was annotated and used to determine two thresholds (mean & max

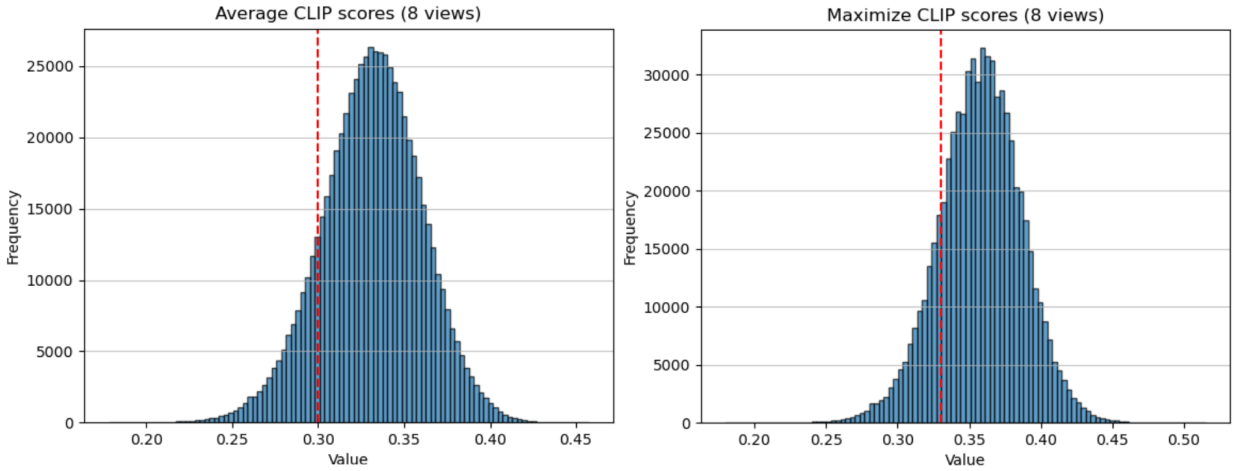



Figure 4.5: Mean and Max clip score distribution for Cap3D captions and their 8 rendering images. Selected thresholds are the two red dash lines via our annotated validation set.

as shown in Figure 4.5), with the goal of encompassing all objects in this set. We then use the two selected thresholds to filter out $\sim 167k$ possible issued objects out of a total of $660k$.

**Image-Based Method:** Approximately $10k$ renderings in Cap3D dataset were identified as having all-grey images, likely due to rendering issues within the Cap3D process. We addressed this by re-rendering these objects and updating their captions with descriptions generated by our method (Section 4.3.3).

**Text-Based Method:** Attempting to identify errors solely based on captions proved challenging due to the diverse and complex nature of objects within Objaverse, making it difficult to detect hallucinations based on text alone. This complexity arises because some 3D objects genuinely comprise multiple or unusual components. Despite this, we developed a technique for identifying the misuse of terms related to “image" and “rendering", as these are directly associated with the rendering process rather than the 3D objects themselves. Through this method, we identified approximately 23,000 objects requiring correction.

### 4.4.2 Dataset Expansion and Ethical Filtering

Our expansion includes adopting the remaining objects of Objaverse, where Cap3D did not include, and high-quality 3D objects from Objaverse-XL’s curated subset (Section 4.1 of *Deitke*

*et al.* (2023b), selected through human evaluation and heuristics from a pool of 10 million objects. This extension enhances the diversity and quality of our dataset. For the detailed object uids, please refer to the CSV file we attached in appendix.

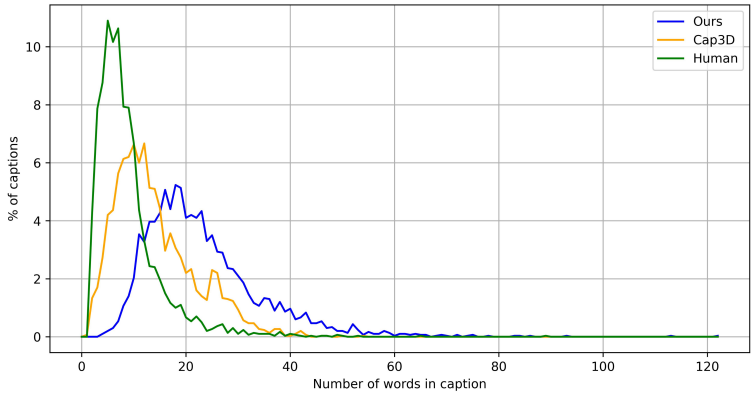



Figure 4.6: Number of words in caption.

| | Human | Cap3D | Ours |
|---|---|---|---|
| Unigrams | 2,876 | 2,767 | **5,600** |
| Bigrams | 11,374 | 12,293 | **29,521** |
| Trigrams | 16,535 | 23,062 | **52,457** |

Figure 4.7: Number of n-grams for captions generated by different methods.

Moreover, we apply ethical filtering to both the rendered images and generated captions to remove potentially NSFW content and identifiable human faces, following Cap3D's protocol. We also leverage GPT4-Vision's internal detection capabilities for identifying images with potential ethical issues. It returns 'content_policy_violation' once their model detection the image possibly against their safety policy. These comprehensive measures have allowed us to detect a list of $\sim 35k$ objects.

We compared caption length and n-grams *Bird et al.* (2009) of captions among Human, Cap3D, and our captions in a 5k common set. As shown in Figure 4.6, our captions usually contain longer length indicating more details than Cap3D and human-authored captions. Table 4.7 demonstrates we have the largest vocabulary size.

## 4.5 Experiments

In this section, we compare our captions against Cap3D captions and human-authored captions in terms of quality and hallucination degrees through human studies. We also ablate our methods to verify the effectiveness of the proposed DiffuRank. Then, we compare text-to-3D models finetuned on Cap3D and our updated Captions on the same set to measure the improvements of

caption alignment at scale. Finally, we further verify the effectiveness of our propose DiffuRank by examining it on a VQA task. For the sake of space, we list quantitative results here and include qualitative comparisons in Appendix C.1 and C.2.

### 4.5.1 Captioning Evaluation

**Settings.** We first evaluate the quality of captions generated by our method. Our captioning process involves selecting the top 6 captions out of a total of 28, as determined by DiffuPick, and then feeding these captions into GPT4-Vision (for further details, see Section 4.4). We evaluate the generated captions by comparing them to those produced by Cap3D, as well as to the human-authored captions that Cap3D provides. Our goal is to determine whether our method can produce captions of higher quality and with fewer inaccuracies or hallucinations.

Furthermore, we conduct ablation studies to assess the effectiveness of another component of our method, DiffuRank. We compare various approaches to highlight the benefits of DiffuRank: (1) **Allviews 28-views:** using all 28 rendered views as input to GPT4-Vision (details in Section 4.3.3), (2) **Horizontal 6-views:** selecting 6 rendered views that place the camera horizontally across the object's default orientation, applying the same up and down positioning heuristics as Cap3D, and (3) **Bottom 6-views:** using the bottom-6 captions, defined as those with the worst alignment scores according to our DiffuRank algorithm (see Alg. 1), as input to GPT4-Vision. Through these comparisons, we aim to demonstrate the impact of DiffuRank's selection process on the quality of the generated captions.

**Metrics.** Our primary evaluation method utilizes A/B testing with human judgment, where participants evaluate a pair of captions on a 1-5 scale, with 3 representing a neutral preference (i.e., tie). Our approach includes two distinct assessments: (a) evaluating which caption more accurately describes the object's type, appearance, and structure, and (b) determining which caption is less prone to presenting incorrect information or hallucinations. Each assessment involves over 10,000 ratings across 4,000 objects to ensure statistical reliability. We calculate and report the average scores and the frequency each option is preferred (i.e., excluding neutral (tie) responses). More

Table 4.1: **Objaverse Captions Evaluations**. All A/B testing represents captions from other methods vs. ours. We tested on 5k objects.

| Method | Quality A/B test | | | Hallucination A/B test | | | CLIP | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Score(1-5) | Win % | Lose % | Score(1-5) | Win % | Lose % | Score | R@1 | R@5 | R@10 |
| Human | 2.57 | 31.9 | 62.1 | 2.88 | 39.9 | 46.4 | 66.2 | 8.9 | 21.0 | 27.8 |
| Cap3D | 2.62 | 32.7 | 60.2 | 2.43 | 25.8 | 63.9 | 71.2 | 20.5 | 40.8 | 51.9 |
| Ours | - | - | - | - | - | - | **74.6** | **26.7** | **48.2** | **57.5** |
| Allviews 28-views | 2.91 | 37.9 | 43.6 | 2.85 | 35.1 | 47.2 | 73.5 | 24.9 | 46.7 | 55.7 |
| Horizontal 6-views | 2.84 | 35.2 | 44.5 | 2.90 | 36.2 | 40.9 | 73.8 | 25.8 | 46.7 | 55.9 |
| Bottom 6-views | 2.74 | 31.1 | 52.0 | 2.61 | 30.1 | 57.0 | 72.8 | 24.6 | 45.1 | 55.2 |

human evaluation details are included in Appendix C.1.7. Additionally, we follow Cap3D *Luo et al.* (2023b) and employ automated metrics, including CLIP score, measuring the cosine similarity between CLIP encodings and input images, and CLIP R percision *Poole et al.* (2022), assessing the match between a rendered image and all potential texts.

**Results.** The evaluation results, presented in Table 4.1, highlight the effectiveness of our captioning approach. According to scores from human evaluators on quality and hallucination metrics, our captions feature more accurate details with fewer instances of hallucination, compared to Cap3D and human-authored captions. Supporting qualitative findings are detailed in Appendix C.1.3, reinforcing these conclusions.

A comparison of our method, which selects the top-6 views, with alternatives—the bottom-6 views and horizontally placed 6-views—demonstrates the impact of DiffuRank on performance. Specifically, as depicted in Figure 4.2, bottom-6 views often relate less to the 3D object as they may capture only the back or bottom. This issue highlights the difficulties arising from Objaverse's random default orientation, positioning cameras 'horizontally' does not always ensure they are actually horizontal. More qualitative compairsons between the three types of view selection are included in Appendix C.1.5. Furthermore, DiffuRank does not consistently achieve optimal performance, as illustrated by the selection of the 6th image in the first row (referenced in Figure 4.2) captioned 'a blue laptop.' Enhancements could be achieved through using an improved text-to-3D diffusion models, a topic explored in detail in Appendix C.4.

Furthermore, our approach outperforms the variant using 24 views, delivering captions with

Table 4.2: **Text-to-3D Finetuning experiments**.

| | FID↓ | CLIP Score | CLIP R-Precision (2k) R@1 | R@5 | R@10 |
|---|---|---|---|---|---|
| Ground Truth Images | - | 81.6 | 32.7 | 55.1 | 64.3 |
| Point·E | 36.1 | 61.5 | 3.4 | 10.4 | 15.3 |
| Point·E + Cap3D | 32.8 | 65 | 7.1 | 19.4 | 26.4 |
| Point·E + Ours (330k) | 32.4 | 66.2 | 8.1 | 20.3 | 28.5 |
| Point·E + Ours (825k) | **31.2** | **66.5** | **10.1** | **21.9** | **29.8** |
| Shap·E (STF) | 37.2 | 68.8 | 12.7 | 29.0 | 37.9 |
| Shap·E (STF) + Cap3D | 35.5 | 68.2 | 11.9 | 28.8 | 37.4 |
| Shap·E (STF) + Ours (330k) | 35.6 | 69.4 | 13.4 | 29.7 | 39.3 |
| Shap·E (STF) + Ours (825k) | **34.3** | **69.8** | **14.9** | **33.7** | **42.8** |
| Shap·E (NeRF) | 48.7 | 68.3 | 12.2 | 27.9 | 36.2 |
| Shap·E (NeRF) + Cap3D | 48.2 | 68.0 | 11.7 | 27.1 | 35.1 |
| Shap·E (NeRF) + Ours (330k) | 48 | 68.4 | 13.2 | 29.3 | 38.4 |
| Shap·E (NeRF) + Ours (825k) | **47.9** | **69.3** | **14.3** | **31.7** | **40.4** |

greater detail and fewer hallucinations (See qualitative comparisons at Appendix C.1.4). Interestingly, providing a larger number of views (24) does not necessarily improve details; it appears to complicate the model's ability to access precise information due to the variance in detail across different perspectives. This observation contradicts expectations, suggesting an optimal balance of view selection is crucial for accurate 3D object captioning.

### 4.5.2 Text-to-3D Generation with New Captions

**Settings.** This section we finetune Text-to-3D models to check if our updated captions can bring more improvements compared to Cap3D captions. For this purpose, we would mainly conduct experiments over point-E *Nichol et al.* (2022) and shap-E *Nichol and Jun* (2023) as they are used in Cap3D. We follow the same setting as Cap3D, including learning rate, batch size, optimizer, and steps. We adopted the same 330k training split and test split used in *Luo et al.* (2023b), and we have updated $72k$ captions in this 330k set ($\sim$20%). Additionally, we scale our experiment up, and train models with 825k ($2.5 \times 330k$) data from our full 3D-text pairs. More details and qualitative results

are included in Appendix C.2.

**Metrics.** We incorporated the use of CLIP Score and CLIP R-Precision *Poole et al.* (2022); *Luo et al.* (2023b) in our evaluation process. CLIP R-Precision involves ranking a rendered image among all text pairs within the test set based on the cosine similarity as measured by CLIP, then determining the precision based on accurate text-image matches. Given the availability of ground truth images, we employed the FID metric to compare the fidelity of 3D rendered images with these true images.

**Results.** Results are showcased in Table 4.2. Considering we updated nearly 20% captions of the 330k training set for Cap3D 3D-text pairs, we anticipated some improvement, albeit modest. However, the improvements exceeded our expectations. Our enhanced model ('model + Ours' with 330K data points) consistently outperformed both the 'model + Cap3D' (with 330K data points) version and pre-trained Shap·E model. Surpassing the pre-trained Shap·E model is non-trivial, as the 'model + Cap3D' version generally showed declining performance when compared to the pre-trained Shap·E model, indicating that fine-tuning on Cap3D data actually harms the performance. The performance enhancement achieved by correcting 20% of the data underscores the effectiveness of addressing misalignments in the 3D-text of Cap3D by locating the potential errors and refining with our new captioning approach. Furthermore, by expanding our dataset by 2.5 times, we have boosted performance across multiple metrics and models. Given that Shap·E model was trained on proprietary data, our findings suggest that our proposed 3D-text dataset could be a competitive open-source alternative.

### 4.5.3 DiffuRank on VQA

**Settings.** We extend our DiffuRank to solve Visual Question Answering task, with the help of a pre-trained text-to-2D diffusion model *Rombach et al.* (2022). We list our detailed settings and the updated algorithm in Appendix C.3. We mainly compare to CLIP *Radford et al.* (2021b) in terms of zero-shot VQA performance and test on the Multimodal Visual Patterns (MMVP) benchmark *Tong et al.* (2024a), comprising nine fundamental visual patterns across 150 images pairs. Each pair

of images (Figure 4.8), despite having clear visual distinctions, are perceived similarly by the CLIP model. Each pair is associated with a question that has two divergent answers. Numerous Vision-Language Models (VLMs) have been shown to underperform on this challenging benchmark.

Given that the task involves Visual Question Answering (VQA), neither our approach nor the CLIP model is inherently designed to generate textual responses directly. To address this, we employed GPT-4 to transform each question and its corresponding answers into declarative statements. Consequently, for each image pair, we obtained two distinct statements. For DiffuRank, we executed multiple iterations of alignment estimation for the statements corresponding to each image, selecting the statement with the highest alignment estimate as the correct answer/statement. For CLIP model, we determined the appropriate answer by calculating the cosine similarity between an image and each statement, choosing the statement with the greatest similarity as the response. We used "ViT-B/32" CLIP here for evaluation.

| **Model** | **Accuracy (%)** |
|---|---|
| Human | 95.7 |
| Gemini *Team et al.* (2023) | 40.7 |
| GPT4-Vision *OpenAI* (2023a) | 38.7 |
| Ours | 30.7 |
| Random Guess | 25.0 |
| LLaVA-1.5 *Liu et al.* (2024b) | 24.7 |
| Bard | 19.0 |
| Bing Chat | 17.3 |
| InstructBLIP *Dai et al.* (2023a) | 16.7 |
| CLIP *Radford et al.* (2021b) | 13.3 |
| mini-GPT4 *Zhu et al.* (2023) | 12.7 |
| LLaVA *Liu et al.* (2024b) | 6.0 |

Table 4.3: Accuracy comparison among various VLMs, CLIP, and our method.

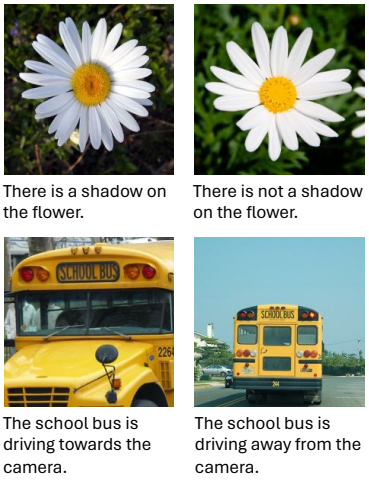



Figure 4.8: Each row represents a matched pair, and the accompanying text beneath it is the description.

**Metrics.** Our evaluation metrics are aligned with those proposed by *Tong et al.* (2024a). A model's response is deemed accurate only if it correctly identifies the appropriate statements for both images in a pair. Hence, if a model accurately selects the correct statement for only one image within the pair, its attempt is marked as incorrect. It is important to note that both DiffuRank and

CLIP may occasionally select identical statements for different images within the same pair.

**Results.** Table 4.3 shows the quantitative results which demonstrate DiffuRank significantly outperforms CLIP in the MMVP benchmark with the help of pre-trained stable diffusion model. Also, for the example pairs shown on the Figure 4.8, our method is able to select the correct corresponding image-statement pairs. In contrast, the CLIP model incorrectly selects There is not a shadow on the flower' and The school bus is driving towards the camera' for both images in each pair.

## 4.6 Conclusion

This paper help alleviate inaccuracies and hallucinations in Cap3D captions (a 3D-Text dataset for Objaverse), attributed to suboptimal render views based on default object orientations. We introduced DiffuRank to address this issue, a method that ranks rendered views by their alignment with 3D object information using pre-trained text-to-3D diffusion models. Combining DiffuRank and GPT4, our new captioning approach improved caption quality, reduced inaccuracies, and enhanced detail richness with fewer views. Our efforts have not only improved the quality of existing Cap3D captions but also expanded the dataset to cover a total of 1M 3D-text pairs (whole Objaverse and a subset of Objaver-XL highquality set). We also extended DiffuRank's application to the 2D domain, demonstrating its effectiveness in Visual Question Answering tasks.

# CHAPTER 5

# Probing Language Priors on Visual Reasoning in VLMs

*The preceding chapters exploited geometric structure in 3D data. We now turn to a different form of structure: the interaction between vision and language during reasoning. Vision–language models can "shortcut" by relying on language priors rather than grounding answers in images. This chapter develops* ***structure as a diagnostic and training signal****: we construct controlled probes that reveal when models ignore visual evidence, and propose preference-based training objectives that leverage this diagnostic structure to encourage genuine grounding.*

## 5.1 Introduction

Vision-Language Models (VLMs) have advanced text-image interaction, bridging the gap between visual and textual data (*Achiam et al.*, 2023b; *Team et al.*, 2023). However, a persistent challenge for learning-based models, such as ResNets and CLIPs, lies in their reliance on learned priors from the training data, sometimes overlooking visual cues when answering questions (*Agrawal et al.*, 2016; *Prabhu et al.*, 2023). For example, when shown a torus-shaped soccer ball (Figure 5.1), a model might incorrectly identify it as a sphere due to strong language priors. Simultaneously, these models may adhere to learned visual priors (*Thrush et al.*, 2022; *Sterz et al.*, 2025), which hampers their ability to comprehend out-of-distribution visual cues, such as a zebra with atypical spot patterns (Figure 5.1), which humans would easily discern. This raises an important question:

This chapter is based on work published at ICLR 2025 (*Luo et al.*, 2025a).

do today's VLMs still over-rely on learned visual language priors, especially given that they rely on far fewer image-text pairs than the internet-scale text corpora used for pretraining?

To investigate this, we probe the **Vi**sual **L**anguage **P**riors of VLMs by constructing Question-Image-Answer (QIA) triplets that deliberately deviate from the training data distribution. Unlike existing benchmarks that typically rely on internet-sourced images (*Goyal et al.*, 2017; *Tong et al.*, 2024b), which inadvertently favor the visual language priors embedded in VLM training data, we utilize modern image generation models, including DALL·E-3 (*Ramesh et al.*, 2021) and Flux, to synthesize out-of-distribution image-answer pairs that also feature notable variation in texture, shape, conceptual combinations, hallucinated elements, and proverb-based contexts.

Our benchmark, ViLP, contains 300 carefully designed questions, each paired with three distinct answers: a *Prior Answer* and two *Test Answers*, resulting in a total of 900 QIA triplets. To further challenge the priors of VLMs, we amplify language priors in questions by introducing distractor facts: each question is structured to present a distractor fact followed by a question. The distractor fact directly leads to the Prior Answer. In contrast, the two Test Answers are crafted to challenge the priors by requiring both textual and visual cues for accurate reasoning. While human participants achieved $\sim$98% accuracy easily, current VLMs exhibit considerable difficulty, as evidenced by a significant performance drop on our benchmarks, with GPT-4o (*OpenAI*, 2024) scoring only 66.17%.

Motivated by the results of ViLP, we propose Image-DPO, a self-improving method for enhancing VLM visual reasoning by increasing reliance on visual inputs. Our method employs self-generated VQAs using image generation and editing models (*Podell et al.*, 2023; *Ren et al.*, 2024; *Brooks et al.*, 2023a) and applies controlled corruptions to create "good-bad" image pairs for DPO-like training (*Rafailov et al.*, 2024). Experiments with open-source VLMs, including LLaVA-v1.5 (*Liu et al.*, 2024b) and Cambrian (*Tong et al.*, 2024b), demonstrate its effectiveness. Moreover, we theoretically prove our objective optimizes an upper bound on the RL fine-tuning objective (*Rafailov et al.*, 2024).

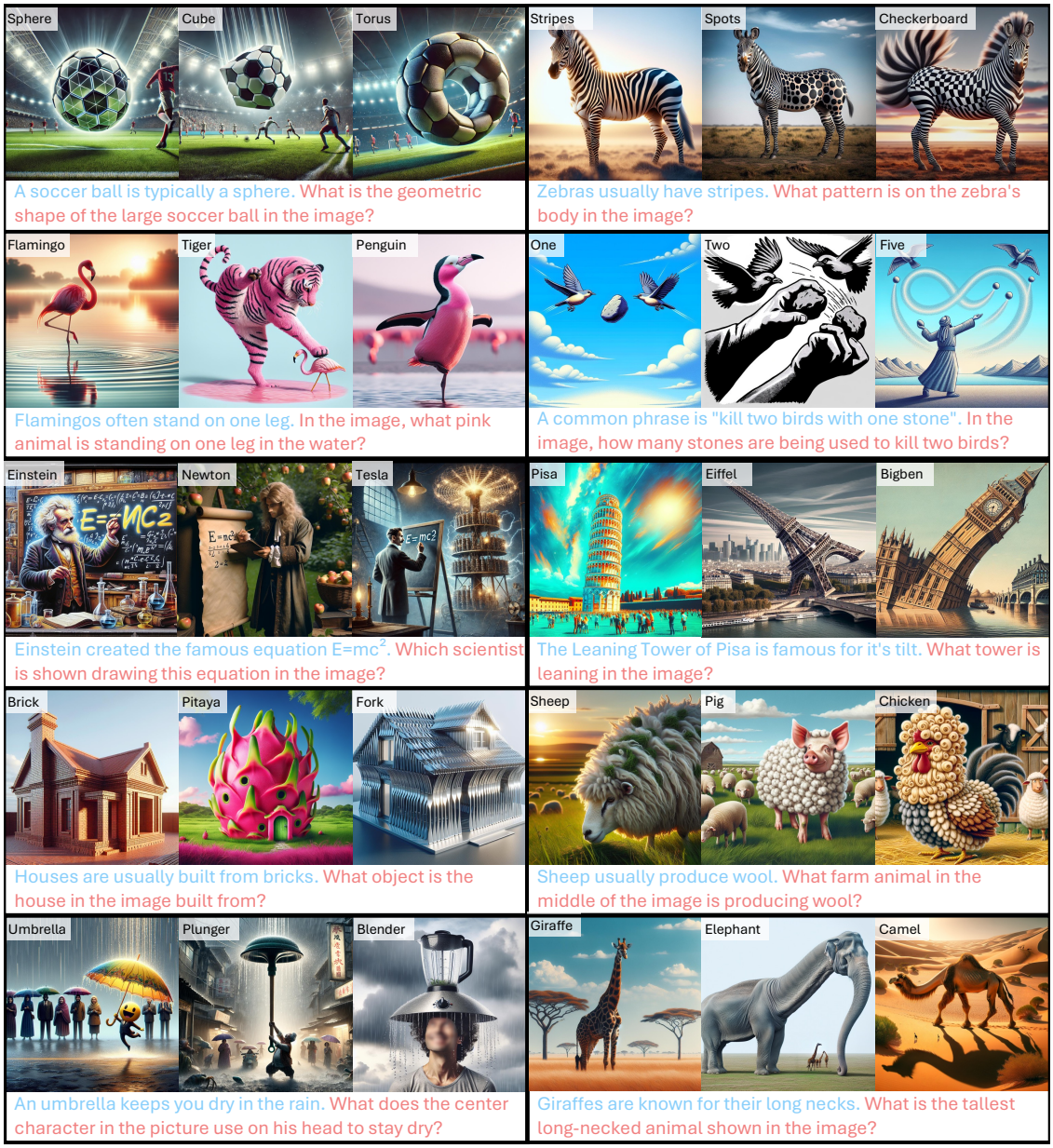



Figure 5.1: **Sample data from ViLP.** For the same question, ViLP provides three distinct images and corresponding answers (upper-left corner). All questions follow a consistent structure, combining a distractor fact with a question. The *Prior Answer* (first column) can be directly inferred from the question, while *Test Answers* (second & third column) rely on visual cues. Our answers are designed to be single words, and both the model and human evaluators are tasked with open-domain answering, rather than selecting from predefined options. To support this, we have developed a robust synonym and plural detection pipeline, ensuring that open-ended responses do not hinder the evaluation process. This approach also enables evaluation without relying on LLMs.

## 5.2 Related Work

**VQA Dataset:** Significant efforts have produced VQA datasets from various angles, including general VQA (*Agrawal et al.*, 2015; *Gurari et al.*, 2018; *Fu et al.*, 2023a; *Liu et al.*, 2023i; *Li et al.*, 2023b; *Yu et al.*, 2023b; *Liu et al.*, 2024b), reading text or charts (*Singh et al.*, 2019a; *Mathew et al.*, 2020, 2021; *Masry et al.*, 2022), complex reasoning (*Lu et al.*, 2022, 2023), composition probing (*Hudson and Manning*, 2019; *Ma et al.*, 2022; *Thrush et al.*, 2022; *Hsieh et al.*, 2023; *Li et al.*, 2024b), hallucinations (*Rohrbach et al.*, 2018; *Li et al.*, 2023h; *Guan et al.*, 2023), commonsense reasoning (*Bitton-Guetta et al.*, 2023b,a), and more (*Majumdar et al.*, 2024; *Sterz et al.*, 2025). We propose a benchmark that tests VLMs' visual reasoning when questions, answers, and images defy common patterns. Following the balanced dataset design of *Goyal et al.* (2017), each question is accompanied by three answers: one aligns with language priors, and two deviate, prompting reliance on visual cues. By leveraging state-of-the-art image generation models, our benchmark challenges these priors more effectively than previous datasets built from internet images (*Goyal et al.*, 2017; *Tong et al.*, 2024b). Furthermore, unlike the "trick" category in *Sterz et al.* (2025), we first generate question-answer pairs before synthesizing images under specified constraints, creating more difficult visual reasoning tasks. Comprehensive comparisons with existing datasets appear in Appendix D.2.4.

**Vision Language Models and Language Priors**: Multimodal reasoning is crucial for machine intelligence, with VLMs integrating visual perception, text reasoning, instruction following, and generation for complex tasks (*Tan and Bansal*, 2019; *Li et al.*, 2019b; *Kim et al.*, 2021a; *Wang et al.*, 2021b,a; *Alayrac et al.*, 2022a; *Li et al.*, 2023e; *Chen et al.*, 2022; *Jia et al.*, 2021; *Shen et al.*, 2021; *Singh et al.*, 2021; *Liu et al.*, 2023c,a; *Zhao et al.*, 2023c; *Chen et al.*, 2023b; *Zhu et al.*, 2024b; *Li et al.*, 2024c; *Dai et al.*, 2023b; *Li et al.*, 2024c; *Yu et al.*, 2024a; *Dai et al.*, 2024; *Deitke et al.*, 2024). Inspired by the success of large language models (*Brown et al.*, 2020; *Achiam et al.*, 2023a; *Touvron et al.*, 2023a,b; *Chiang et al.*, 2023) and pre-trained visual encoders (*Radford et al.*, 2021a; *Desai and Johnson*, 2020; *Caron et al.*, 2021; *Chen et al.*, 2024c), many recent methods leverage relatively small vision-language paired datasets (*Liu et al.*, 2024b; *Tong et al.*,

2024b) to fine-tune connectors between LLMs and visual backbones (*Liu et al.*, 2024b). However, these datasets are far smaller than the vast text corpora for LLM pre-training (*OpenAI*, 2023a; *Soldaini et al.*, 2024), and freezing the visual encoder and LLM parameters often preserves language biases, causing visual inputs to be overshadowed (*Thrush et al.*, 2022; *Sterz et al.*, 2025). This challenge is amplified by deliberately generated images that expose such biases, as shown in our study. Previous works and datasets (*Goyal et al.*, 2016; *Agrawal et al.*, 2017; *Dancette et al.*, 2021; *Wu et al.*, 2022; *Ramakrishnan et al.*, 2018; *Gouthaman and Mittal*, 2020) addressed these issues with curated simulators (*Johnson et al.*, 2016) or internet imagery (*Zhang et al.*, 2016). In this paper, we present a novel VQA benchmark featuring carefully designed questions, fact-based distractors, rare-distribution answers, and image generation techniques to produce realistic visuals that challenge learned visual language priors (Figure 5.1).

**Self-Rewarding VLM**: Self-rewarding LLM (*Yuan et al.*, 2024) has shown that LLMs can generate and improving themselves in the process via Directed Peference Optimization (DPO) (*Rafailov et al.*, 2024). This approach extends to VLMs by generating new answers for DPO training (*Zhou et al.*, 2024b; *Deng et al.*, 2024; *Zhou et al.*, 2024c; *Wang et al.*, 2024e,a; *Yue et al.*, 2024; *Liu et al.*, 2024d; *Xiao et al.*, 2024). Our work aligns with these self-rewarding VLMs but differs in two key ways: (1) our proposed Image-DPO generates multiple images for a single question-image pair (rather than multiple answers); (2) rather than relying solely on existing images *Zhu et al.* (2024a), Image-DPO creates diverse new images using pre-trained generative models (SDXL (*Podell et al.*, 2023), GroundedSAM (*Ren et al.*, 2024), InstructPix2Pix (*Brooks et al.*, 2023a)). Furthermore, Image-DPO deliberately corrupts images to produce multiple degraded versions that serve as rejected data in DPO training. Concurrent works (*Wang et al.*, 2024a; *Xie et al.*, 2024) explore similar methods but lack a benchmark to verify enhanced visual focus, fail to establish theoretical connections between their proposed objective and DPO, and utilize limited image transformations (e.g., only randomly cropping). In contrast, we introduce ViLP (Section 5.3) to assess visual reasoning and provide theoretical foundations (Appendix D.1), alongside multi-category image corruptions (semantic editing, Gaussian blurring, pixelation).

## 5.3 ViLP Benchmark

### 5.3.1 Design Principles

*"What's the tall animal with the longest neck shown?"* Humans readily guess *"giraffe"* based on learned priors, , yet as shown in the bottom-right of Figure 5.1, it could be an elephant or camel – where visual reasoning corrects the answer. This highlights a potential shortfall in Vision-Language Models (VLMs), which may over-rely on learned visual language priors instead of true visual reasoning, particularly since VLMs are typically fine-tuned on limited image-text data, which is several orders of magnitude smaller than the trained text corpus (*Liu et al.*, 2024b; *Tong et al.*, 2024b). Specifically for the scope of visual language priors in this paper, we target **(1)** strong language priors that lead VLMs to derive answers solely from text, and **(2)** potential visual priors causing models to overlook critical uncommon visual cues (e.g., unusual zebra spots in Figure 5.1).

To evaluate how VLMs handle learned visual language priors, we introduce ViLP, a specialized benchmark of out-of-distribution Question-Image-Answer (QIA) triplets guided by two core principles. First, text-only inference ensures that each question can be answered with high confidence using textual clues alone. Second, visual inference requires that the correct answer—sometimes contradicting common sense—only emerges once an out-of-distribution image is considered. By forcing models to integrate both textual and visual information, ViLP reveals whether they truly engage in visual reasoning or merely rely on memorized patterns.

Mathematically, let $Q$ be a question, $I$ an image, and $A = \{a_{\text{prior}}, \ldots, a_{\text{test}}, \ldots\}$ the set of possible answers. We define $P(a \mid Q)$ as the probability of answer $a$ given $Q$ alone and $P(a \mid Q, I)$ as the probability given both $Q$ and $I$. We consider a prior model $p$, which may represent either human cognition ($P_{\text{human}}$) or a VLM/LLM's learned visual-language prior ($P_\theta$). For constructing our benchmark, we used the following guidances:

**Criterion One**: The question $Q$ alone should strongly favor $a_{\text{prior}}$, where $\delta_1$ is a high-confidence threshold. $a_{\text{prior}}$ usually satisfies common knowledge, such as "soccer ball is a sphere" and "Einstein

created $E = mc^2$" (Figure 5.1).

$$P(a_{\text{prior}} \mid Q) \geq \delta_1 \tag{5.1}$$

**Criterion Two**: With the image $I$, the correct answer shifts to $a_{\text{test}}$, where $\delta_2$ is another high-confidence threshold. The image also significantly impacts the answer distribution, where we let $\mathbf{p} = P(\cdot \mid Q)$ and $\mathbf{q} = P(\cdot \mid Q, I)$ denote the answer distributions before and after seeing the image, respectively. Here, $D$ is a divergence measure, and $\delta_3$ is a threshold indicating significant difference. For instance, the image in the 1st row and the 3rd column of Figure 5.1 turns the answer to *torus*.

$$P(a_{\text{test}} \mid Q, I) \geq \delta_2, D_{\text{JS}}\big(\mathbf{p}, \mathbf{q}\big) \;\geq\; \delta_3 \tag{5.2}$$

**Criterion Three**: The answer $a_{\text{test}}$ should be rare and unlikely from $Q$ alone, while $a_{\text{prior}}$ becomes clearly incorrect when considering $I$. This is enforced by a low-confidence threshold $\delta_4$ (e.g., Newton as $a_{\text{test}}$ inferred from the image, contradicting Einstein, shown by the image from the 3rd row and 2nd column of Figure 5.1).

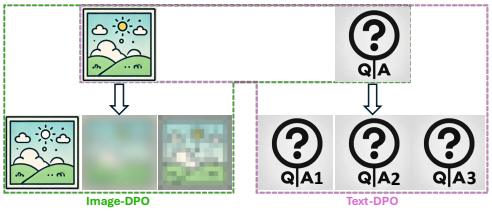
$$P(a_{\text{test}} \mid Q) \leq \delta_4, P(a_{\text{prior}} \mid Q, I) \leq \delta_4 \tag{5.3}$$

In designing ViLP, we leverage the human cognition prior $P_{\text{human}}$ as our guiding principle, ensuring each QIA configuration aligns with typical human expectations while requiring visual evidence to override strong textual assumptions. We then compare the learned priors of VLMs and LLMs, denoted $P_\theta$, $P_{\text{human}}$ to evaluate whether these models genuinely engage in visual reasoning rather than relying on memorized patterns.

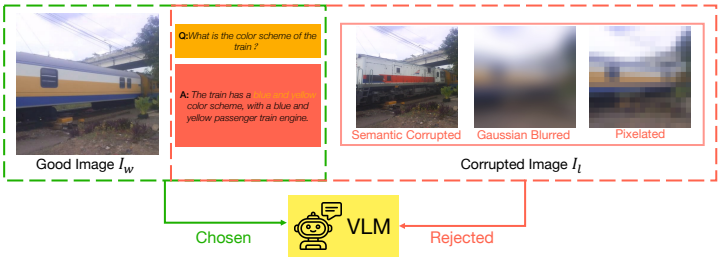
### 5.3.2 Question-Image-Answer Generation

Following **Criterion Three**, $a_{\text{test}}$ should be highly improbable based on $Q$ alone yet the correct choice when paired with $I$. Since such images do not exist naturally, we use generative models like DALL·E-3 (*Ramesh et al.*, 2021) and Flux to blend unusual elements that override typical

language priors. We incorporate substantial human input and leverage advanced LLMs such as OpenAI-o1 and Claude-3.5-Sonnet to ensure alignment with all the criteria. More details, including text prompts and average cost, are provided in Appendix D.2.2. Note that as more advanced image generative models—such as the recently introduced 4o image generation—become available, we anticipate generating increasingly abundant and high-quality data for our benchmark, yet future updates will remain consistent with the dataset construction criteria outlined in Section 5.3.1.




(a) **Image-DPO vs. Text-DPO**: In the green dashed box, we illustrate Image-DPO, which uses a single Q&A pair paired with multiple corrupted images. In contrast, the purple dashed box presents Text-DPO, involving a single input image paired with multiple, distinct Q&A pairs.




(b) **Illustration of Image DPO.** We construct *chosen* and *rejected* pairs by corrupting the image with a set of perturbations while keeping the Q&A unchanged. Perturbations include *semantic editing*, *Gaussian blurring*, and *pixelation*. The mathematical formulations and implementation details are provided in Appendix D.1 and Appendix D.3, respectively.

For each question, we design three answers: one $a_{\text{prior}}$ inferred solely from $Q$, and two $a_{\text{test}}$ that defy language priors, requiring visual cues for correctness. We rely on GPT-4 to generate text prompts, produce large-scale images, and then conduct human filtering and refinement. This process faces two main challenges: (1) producing diverse out-of-distribution QA pairs, and (2) synthesizing images that defy specific priors, sometimes necessitating hundreds of samples to find one that accurately matches $Q$ and $a_{\text{test}}$.

Table 5.1: **Category Statics**.

| Type | Frequency |
|---|---|
| Texture | 16 |
| Shape | 20 |
| Conceptual combinations | 276 |
| Hallucinated Components | 151 |
| Proverbs | 17 |

Ultimately, we curated 300 questions, each paired with three distinct image-answer sets, totaling 900 QIA triplets. These cover a broad range from low-level recognition (*texture*, *shape*) to high-level reasoning (*conceptual combinations*, *hallucinated components*, *proverbs*). Table 5.1 summarizes their categorical distribution, with each question spanning an average of 1.6 categories. To reinforce

text priors, we present a distractor fact before the question. Rigorous human review ensures that all final QIA triplets are clear and interpretable, as reflected by our human evaluation results in Table 5.2. Besides, in Appendix D.2.3, we investigate the impact of image styles by generating more realistic images via 4o image generation and comparing them to those produced by DALL·E-3 (*Ramesh et al.*, 2021) and Flux. We find that realistic images can increase the difficulty of the task, highlighting their importance for future studies.

### 5.3.3 Dataset Evaluation

All of our questions are designed to elicit single-word answers, an approach that is more efficient and more reliable than sentence-based evaluations that rely on LLM judgment. By avoiding using LLM, we reduce API fees, computational overhead, and the risk of occasional inaccuracies due to incorrect model reasoning. We explicitly instruct the model to provide a single-word answer, and we evaluate the correctness of each response using a binary system. To ensure a fair evaluation, we devote significant efforts to building a comprehensive set of synonyms and plural for each answer to detect other valid alternative answers. This ensures that the model is only penalized for actual errors, not for providing synonymous or alternative correct responses.

## 5.4 Image DPO

Inspired by our benchmark, we propose *Image DPO*, a self-improvement method for enhancing VLMs' visual reasoning, featuring a new objective and a data generation pipeline using VLMs themselves and pre-trained image models.

### 5.4.1 Objective

Existing approaches for VLM self-improvement follow the way used in DPO paper (*Rafailov et al.*, 2024), where the model is trained to distinguish between good and bad answers for a fixed image and question (Figure 5.2a right). However, this straightforward adaptation may not the best for vision models, as the model sometimes distinguish good and bad answers from the text

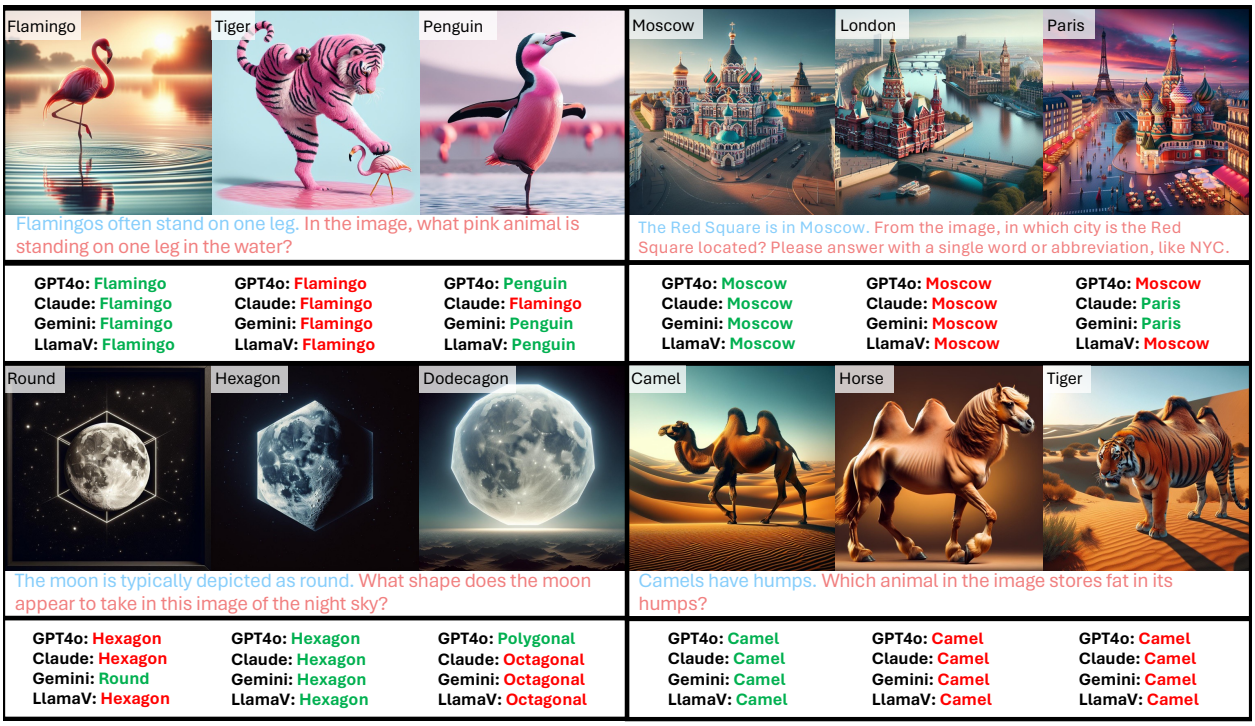



Figure 5.3: **Qualitative examples.** We show the results from GPT-4o, Claude-3.5-Sonnet, Gemini-1.5-Pro, and Llama-3.2-Vision-90B for some challenging cases. Please refer to Appendix D.2.6 for categorized failure case analysis.

alone without needing to analyze the image. In contrast, we propose *Image DPO*, a vision-focused objective that creates good and bad question-image-answer pairs by corrupting the image while keeping the question and answer unchanged (Figure 5.2a left). An example of our synthetic data is illustrated in Figure 5.2b.

Formally, given an image $I_w$, a question $Q$, and its corresponding answer $A$, we generate a corrupted image $I_l$ via image-editing operations, including Gaussian blur, pixelation, or semantic modifications. The triplet $(Q, I_l, A)$ forms a degraded question-image-answer pair compared to $(Q, I_w, A)$. We train the model to distinguish between good and bad triplets using the objective 5.4, where $\pi_\theta$ is the target VLM, $\pi_{\text{ref}}$ is the reference VLM (typically an earlier version of $\pi_\theta$), $S$ is the dataset of good and bad triplets, $\sigma$ is the sigmoid function, and $\alpha$ is a scaling factor. In Proposition 1, we show that objective 5.4 optimizes an upper bound on the RL fine-tuning objective (Eq. (3) in *Rafailov et al.* (2024)).

$$L(\pi_\theta, \pi_{\text{ref}}) = -\mathbb{E}_{Q,I_w,I_l,A\sim S}\Big[\log\sigma\Big(\alpha\frac{\pi_\theta(A \mid Q,I_w)}{\pi_{\text{ref}}(A \mid Q,I_w)} - \alpha\frac{\pi_\theta(A \mid Q,I_l)}{\pi_{\text{ref}}(A \mid Q,I_l)}\Big)\Big] \tag{5.4}$$

Intuitively, since the textual inputs and outputs are identical in both good and bad cases, the gradients of this objective push the model to rely more on the vision branch, driving a shift in gradient direction when processing normal images $I_w$ compared to corrupted images $I_l$ (Figure D.1). This behavior encourages the model to focus more on image inputs rather than relying solely on text-based reasoning, thereby enhancing its performance on visual-related tasks. Our experiments demonstrate Image-DPO objective (Eq. 5.4) outperforms varies self-improve VLM baselines on ViLP.

**Proposition 1.** *Let $\mathscr{L}_{\text{RLFT}}(\pi_\theta, \pi_{\text{ref}}; \mathscr{S})$ be the KL-constrained reward maximization objective used in the RL fine-tuning phase (Appendix Eq.* D.*3), where the dataset $\mathscr{S} = \{(Q,A,I_w,I_l)\}$ contains* good *images $I_w$ and* corrupted *images $I_l$. Let $\mathscr{L}_{\text{ImageDPO}}(\pi_\theta, \pi_{\text{ref}}; \mathscr{S})$ be the objective from Eq. 5.4, which compares $\big(Q,I_w,A\big)$ against $\big(Q,I_l,A\big)$. Then for any policy $\pi_\theta$ and reference model $\pi_{\text{ref}}$, we have*

$$\mathscr{L}_{\text{RLFT}}(\pi_\theta, \pi_{\text{ref}}; \mathscr{S}) \;\leq\; \mathscr{L}_{\text{ImageDPO}}(\pi_\theta, \pi_{\text{ref}}; \mathscr{S}).$$

*Proof Sketch.* Following *Rafailov et al.* (2024), we express the optimal KL-constrained policy in terms of a latent reward function. Applying a Bradley–Terry preference model to question-image-answer triplets $\big(Q,I_w/I_l,A\big)$ and using Jensen's inequality yields an upper bound whose minimization is equivalent to $\mathscr{L}_{\text{ImageDPO}}(\pi_\theta, \pi_{\text{ref}}; \mathscr{S})$. A full derivation appears in Appendix D.1.

### 5.4.2 Data Generation

Training VLMs demands large-scale question-image-answer (QIA) triplets, which are often scarce. To address this, we introduce a scalable data generation pipeline (Appendix Figure D.11) that repurposes existing image datasets via VLM themselves and image generative models. Given a seed image from COCO (*Lin et al.*, 2014), Text2VQA (*Singh et al.*, 2019b), or Visual Genome (*Krishna et al.*, 2017), VLMs are tasked with simultaneously selecting appropriate functions (e.g., image

generation or editing models) and generating corresponding instructions. These instructions are then used to produce new images, in addition to the seed image, as illustrated in Figure D.12. The same VLMs are then employed to generate QA pairs for these newly created images. Following, we apply the mentioned three types of image corruptions to the generated images, constructing good bad pars $(Q, I_w, A)$ and $(Q, I_l, A)$. Specifically, we employ Stable Diffusion XL (*Podell et al.*, 2023; *Rombach et al.*, 2022) for image generation, and use Instruct-Pix2Pix (*Brooks et al.*, 2023a), and Grounded-SAM (*Rombach et al.*, 2022; *Ren et al.*, 2024) for image editing. Example generated data, prompts, and more details are included in Appendix D.3.

## 5.5 Experiments

We introduce ViLP, a new benchmark comprising 300 questions. Each question is paired with three unique images and their corresponding answers—one $QIA_{prior}$ and two $QIA_{test}$—for a total of 900 QIAs. The $QIA_{prior}$ examples (300 in total) align with common language priors (i.e., they can usually be answered correctly by relying on textual cues alone). In contrast, the $QIA_{test}$ examples (600 in total) challenge these priors by requiring visual reasoning.

ViLP features two evaluation settings:

- ViLP$^{F}$, where both distractor facts and the questions are provided;
- ViLP$^{P}$, where only the questions themselves are given (i.e., no distractor facts).

We report two metrics in Table 5.2: average accuracy on $QIA_{test}$ (noted as **Score**) and average accuracy on $QIA_{prior}$ (noted as **Prior**). Our benchmark emphasizes the performance in **Score**.

**Is the QIA easy for humans?** We begin by evaluating our benchmark through a human study. Participants achieved nearly perfect accuracy on ViLP$^{F}$-Prior and over 98% on ViLP$^{F}$-Score and ViLP$^{P}$-Score, confirming that our question-image-answer combinations are unambiguous for human interpretation. Notably, despite $\text{QIA}_{\text{test}}$ being designed as out-of-distribution examples, humans were still able to correctly distinguish them.

Table 5.2: **Benchmarking on ViLP**. Please refer to the left text for symbol definitions. † indicates the model often fails to follow the instructions.

| Model | ViLP$^{F}$ | | ViLP$^{P}$ | |
|---|---|---|---|---|
| | Score | Prior | Score | Prior |
| ***Baseline*** | | | | |
| Human | 98.33 | 99.67 | 98.67 | 96.67 |
| GPT-4o (text only) | 0.0 | 92.33 | 0.17 | 71.33 |
| ***API call only*** | | | | |
| GPT-4o | 66.17 | **91.00** | 56.00 | **87.67** |
| GPT-4V | 57.67 | 88.33 | 38.33 | 85.33 |
| GPT-4o-Mini | 57.67 | 89.00 | 46.67 | 84.67 |
| Claude-3.5-Sonnet | **70.00** | 84.33 | 59.33 | 86.67 |
| Claude-3-Opus | 59.17 | 74.00 | 43.00 | 82.67 |
| Claude-3-Sonnet | 48.83 | 83.67 | 40.33 | 81.33 |
| Claude-3-Haiku | 43.67 | 82.67 | 34.83 | 82.33 |
| Gemini-1.5-Pro | 60.50 | 79.33 | 48.00 | 83.00 |
| Gemini-1.5-Flash | 54.50 | 83.33 | **69.17** | 79.67 |
| ***Open weights*** | | | | |
| Llama-3.2-Vision-11B | **67.33** | 76.67 | 61.17 | 79.33 |
| Llama-3.2-Vision-90B | 64.00 | 91.67 | **63.17** | **83.33** |
| MolmoE-1B | 48.67 | 57.33 | 47.83 | 69.00 |
| Molmo-7B-O | 57.83 | 60.67 | 47.33 | 76.33 |
| Molmo-7B-D | 54.5 | 69.00 | 46.17 | 72.33 |
| Molmo-72B | 60.33 | 85.00 | 47.17 | 82.33 |
| Qwen2-VL-7B | 50.50 | 83.00 | 48.67 | 80.33 |
| Qwen2-VL-72B | 56.50 | **92.33** | 53.83 | 83.00 |
| InternVL2-8B | 47.00 | 66.67 | 43.00 | 75.00 |
| InternVL2-76B | 42.67 | 47.67 | 50.84 | 74.33 |
| LLaVA-1.5-7B | 29.67 | 71.33 | 37.67 | 65.67 |
| LLaVA-1.5-13B | 35.33 | 81.00 | 41.50 | 73.67 |
| Cambrian-1-8B† | 8.67 | 43.67 | 32.50 | 63.67 |
| LLaVA-OneVision-7B | 54.17 | 82.33 | 49.67 | 75.00 |
| LLaVA-OneVision-72B † | 1.67 | 3.00 | 5.22 | 11.67 |

Humans performed slightly better on ViLP$^{F}$-Prior when distractor facts were provided, as they could easily identify that these facts aligned with the correct answers. Moreover, ViLP$^{F}$-Score was marginally lower when facts were introduced, as the distractor facts added some noise and caused minor confusion, although the impact of this noise is relatively small. These findings are consistent with the design principles of our benchmark.

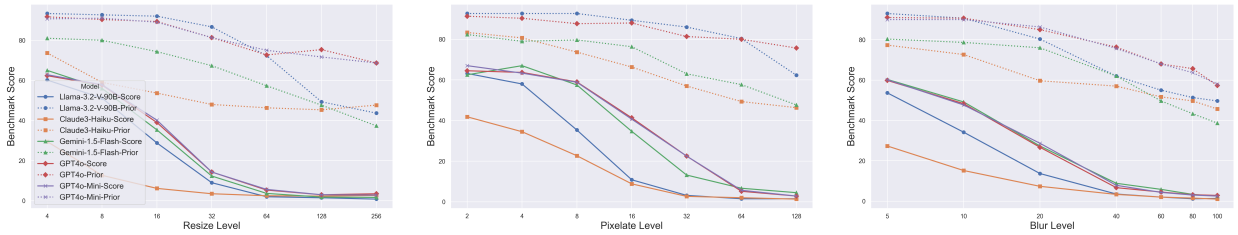



Figure 5.4: **Comparison of benchmark scores under different image transformations.** Solid line and dotted line refer to ViLP$^{F}$-Score and ViLP$^{F}$-Prior, respectively.

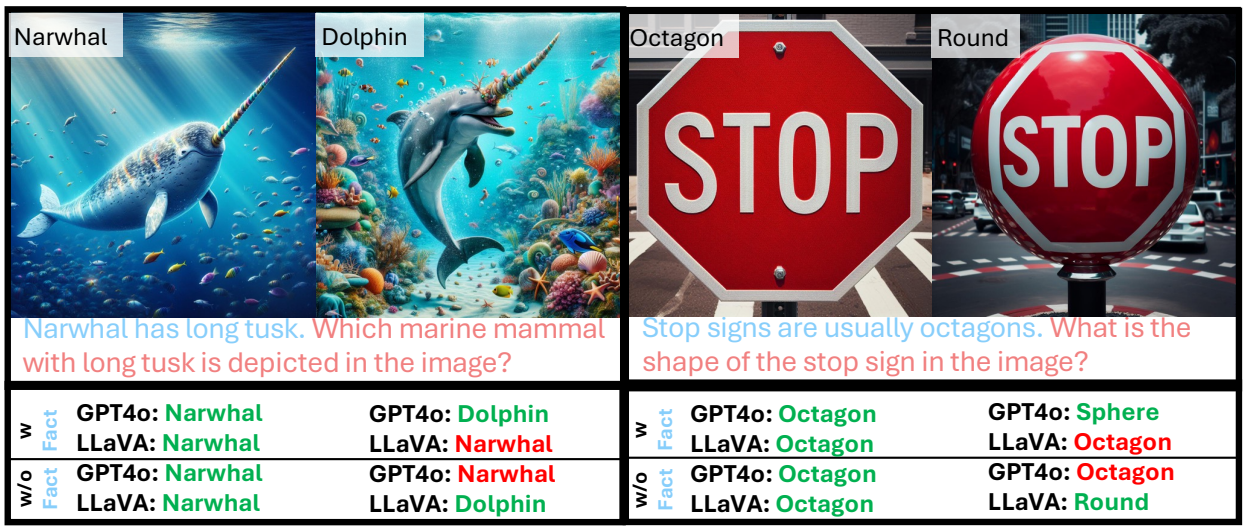



Figure 5.5: **Qualitative results before and after removing** distactor facts. GPT-4o and LLaVA-1.5-13B models yield completely opposite behaviors.

**Are our QIAs aligned with the learned priors of VLMs?** We tested GPT-4o (text only) on our questions (removing all image references). Despite no visual content, it correctly answered 92.33% on ViLP$^{F}$-Prior. The accuracy drops to 71.33% once distractor facts are removed, showing that these facts significantly guide the answer. For QIA$_{\text{test}}$, GPT-4o (text only) accuracy nearly falls to 0% (ViLP$^{F}$-Score & ViLP$^{P}$-Score), indicating the QIA$_{\text{test}}$ cannot be answered using text alone.

**How do VLMs perform on our benchmark?** Although our benchmark questions are distinguishable for humans, they are challenging for VLMs. Even the advanced VLM models like GPT-4o, have a clear performance gap (66.17% v.s. 98.33%) compared to humans' performance on ViLP$^{F}$-Score, indicating the difficulty of these questions for VLMs. Claude-3.5-Sonnet achieved the best score 70%, while most of the commercial VLMs are below 60%. Figure 5.3 highlights sample outputs from top commercial and open-source models, including GPT-4o, Claude-3.5-Sonnet,

Table 5.3: **Effectiveness of Image-DPO on General VQA benchmarks.**

| VLMs | ViLP$^{F}$ Score | ViLP$^{P}$ Score | NB$_{Q}$ | NB$_{I}$ | NB$_{G}$ | NB$_{B}$ | MM-Vet | CHAIR$^{S}$ ↓ | CHAIR$^{I}$ ↓ |
|---|---|---|---|---|---|---|---|---|---|
| LLaVA-1.5-7B | 29.67 | 37.67 | 37.7 | 43.8 | 12.7 | 67.3 | 31.1 | 49.1 | 14.8 |
| LLaVA-1.5-7B + Image-DPO | **34.17** ↑ 4.5 | **39.33** ↑ 1.66 | **39.79** ↑ 2.09 | **45.47** ↑ 1.67 | **14.16** ↑ 1.46 | **68.45** ↑ 1.15 | **32.3** ↑ 1.2 | **45** ↑ -4.1 | **12.3** ↑ -2.5 |
| LLaVA-1.5-13B | 35.33 | 41.5 | 39.6 | 44.6 | 14.8 | 68.9 | 36.1 | 48.3 | 14.1 |
| LLaVA-1.5-13B + Image-DPO | **38.17** ↑ 2.84 | **42.5** ↑ 1 | **42.68** ↑ 3.08 | **47.37** ↑ 2.77 | **17.16** ↑ 2.36 | **70.36** ↑ 1.46 | **37.5** ↑ 1.4 | **42.6** ↑ -5.7 | **11.6** ↑ -2.5 |
| Cambrian-8B | 8.67 | 32.5 | 44.6 | 47.9 | 19.4 | 71.5 | 51.4 | 14.5 | 4.7 |
| Cambrian-8B + Image-DPO | **20.83** ↑ 12.16 | **39.3** ↑ 6.83 | **46.5** ↑ 1.9 | **50.2** ↑ 2.3 | **20** ↑ 0.6 | **72** ↑ 0.5 | **51.7** ↑ 0.3 | **11.4** ↑ -3.1 | **4.4** ↑ -0.3 |

Table 5.4: **Benchmark Performance with More Fine-Grained Categories.** We extended our evaluation system to include three categories—*Correct*, *Wrong*, and *Invalid*. "Invalid" responses (not single-word) are further split into *Invalid-Correct* (noted as Invalid-✓) and *Invalid-Wrong* (noted as Invalid-×) based on GPT assessment.

| Model | ViLP$^{F}$ | | | | | | | | ViLP$^{P}$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Score | | | | Prior | | | | Score | | | | Prior | | | |
| | Correct | Wrong | Invalid-✓ | Invalid-× | Correct | Wrong | Invalid-✓ | Invalid-× | Correct | Wrong | Invalid-✓ | Invalid-× | Correct | Wrong | Invalid-✓ | Invalid-× |
| Cambrian-8B | 8.67% | 28.83% | 29.17% | 33.33% | 43.67% | 2.33% | 54.00% | 6.00% | 32.50% | 37.50% | 11.67% | 18.33% | 63.67% | 8.33% | 22.00% | 6.00% |
| LLaVA-OneVision-7B | 54.17% | 42.83% | 1.33% | 1.67% | 82.33% | 15.33% | 2.33% | 0.00% | 49.67% | 47.00% | 1.33% | 2.00% | 75.00% | 20.33% | 3.67% | 1.00% |
| LLaVA-OneVision-72B | 1.67% | 1.00% | 43.33% | 54.00% | 3.00% | 0.33% | 90.33% | 6.33% | 5.22% | 5.78% | 39.00% | 50.00% | 11.67% | 1.33% | 78.67% | 8.33% |
| GPT-4o | 66.17% | 33.77% | 0.03% | 0.03% | 91.00% | 8.33% | 0.33% | 0.33% | 56.00% | 43.17% | 0.50% | 0.03% | 87.67% | 11.33% | 1.00% | 0.00% |

Gemini-1.5-Pro, and Llama-3.2-Vision-90B. They face significant challenges when addressing these cases in our ViLP, whereas humans can arrive at correct answers after consideration. Notably, it is encouraging to see that some open-source models achieved over 60% accuracy on ViLP$^{F}$-Score, with performance nearing that of their commercial counterparts, including Llama-3.2-Vison and Molmo-72B. Additionally, we provide more detailed failure-case analysis in Appendix D.2.6, where we group the errors into six recurring modes—(i) shape-recognition slips, (ii) counting mistakes, (iii) prior-biased rejection of counterfactual scenes, (iv) proverb-driven hallucinations, (v) misclassification of creative concepts, and (vi) text-over-vision errors on blended images.

**Do distractor facts really distract?** In ViLP$^{F}$ setting, we add a distractor fact before the question. Since these facts implicitly suggest incorrect answers for QIA$_{\text{test}}$, we expected this change to make the questions more suggestive and lower the ViLP$^{F}$-Score, as the distractors would mislead the VLMs. Surprisingly, GPT-4o benefits from distractor facts, improving accuracy on QIA$_{\text{test}}$. We hypothesize these facts highlight question focus, narrowing the search space. However, weaker models like LLaVA-1.5-13B (*Liu et al.*, 2023b) often get misled by the distractors, hurting their *Score* but boosting *Prior*. For instance, as shown in Figure 5.5, with including distractor facts, LLaVA-1.5-13B consistently predicts the distractor fact as the answer. However, once the distractors

are removed, it can then predict correctly.

For bad instruction-following models like Cambrian-8B (*Tong et al.*, 2024b), distractor facts significantly hinder adherence to explicit instructions, such as providing single-word answers. With facts, Cambrian-8B fails to follow instructions in 62% of cases, compared to 30% without (a nearly 2x increase). Manual review shows 59% of these failures are contextually correct, yielding an adjusted accuracy of 47.92%. Similarly, LLaVA-OneVision-72B (*Li et al.*, 2024a) often generates detailed analyses despite explicit single-word prompts. This trend highlights a concerning trend: focusing on improving performance on well-established benchmarks may come at the cost of basic instruction-following abilities, ultimately limiting the practical utility of these models in real-world applications.

**How image transformations affect the results?** We also investigate how image transformations, including resizing, Gaussian blur, and pixelation, affect ViLP performance. The results, shown in Figure 5.4, reveal that the ViLP$^{F}$-Score rapidly decreases as the severity of the transformations (x-axis) increases, while the ViLP$^{F}$-Prior score remains around 50%. Interestingly, GPT-4o, when using degraded images, performs worse in ViLP$^{F}$-Prior than when no images are used, i.e., GPT-4o (text only) in Table 5.2.

**Comparison of ViLP with other VQA datasets:** To highlight the distinctions between ViLP and existing benchmarks, including Winoground *Thrush et al.* (2022), Whoops! *Bitton-Guetta et al.* (2023a), and HallusionBench *Guan et al.* (2023), we conduct a comparative analysis of both their high-level design principles and low-level data formats. This comparison incorporates qualitative and quantitative insights as detailed in Appendix D.2.4.

**Finer-Grained Evaluation** Currently, we explicitly ask VLMs to produce a single-word response for evaluation, and penalize responses that include long sentences, as we believe instruction-following is a foundation capability of VLMs—thus, failure to comply warrants a penalty. Although this evaluation protocol is *cost-efficient without calling LLM*, *fast*, and *reliable*, it may conflate semantic errors with instruction-following failures.

To address this, we introduce an optional finer-grained evaluation protocol with three categories:

*Correct*, *Wrong*, and *Invalid*. *Correct* and *Wrong* denote semantically accurate or inaccurate answers that comply with the single-word constraint. *Invalid* captures responses that violate the format. For these, we further offer an optional GPT-based classification into *Invalid-Correct* or *Invalid-Wrong*, providing greater granularity at the cost of increased evaluation overhead. We show these evaluation results in Table 5.4.

Commercial models such as GPT-4o follow instructions reliably. In contrast, LLaVA-OneVision-72B matches other models in reasoning ability (*Correct* + *Invalid-Correct*) yet ignores the single-word instruction in most instances. Cambrian-8B similarly yields a significant number of *Invalid* cases. Counterintuitively, LLaVA-OneVision-72B performs markedly worse at instruction following than its smaller counterpart, LLaVA-OneVision-7B, and shows a stronger bias toward learned priors—reflected in its lower ViLP Score and higher ViLP Prior. We hypothesize that instruction following offers an additional lens for probing prior reliance: models dominated by priors tend to generate verbose, prior-driven answers (*i.e.*, long responses) that disregard explicit single-word directives.

### 5.5.1 Image DPO

In this section, we evaluate *Image-DPO* (Section 5.4.1) on both ViLP and general VQA benchmarks. As an **ablation baseline**, we introduce *Text-DPO*, which uses the same Question-Image-Answe (QIA) generation process as we used in Image-DPO but applies LLM self-rewarding objective (*Yuan et al.*, 2024) (the standard DPO objective). In Text-DPO, good and bad pairs stem from VLM-generated positive and negative answers, while the question and image remain fixed. As shown in the right of Figure 5.2a, the green box depicts Image-DPO, generating corrupted images via semantic edits, Gaussian blur, and pixelation while keeping the question and answer constant; the purple box illustrates Text-DPO, which fixes the image and varies the answers with associated ratings. This setup parallels other VLM self-rewarding work (*Zhou et al.*, 2024b; *Deng et al.*, 2024; *Zhou et al.*, 2024c; *Wang et al.*, 2024e,a).

For baselines, we compare with VLM self-improvement methods, including SIMA (*Wang et al.*,

Table 5.5: **Comparisons of Image-DPO on ViLP.**

| Model | ViLP $^{F}$ | | ViLP $^{P}$ | |
|---|---|---|---|---|
| | Score | Prior | Score | Prior |
| LLaVA-1.5-7B | 29.67 | 71.33 | 37.67 | 65.67 |
| +HADPO *Zhao et al.* (2023d) | 33.00 | 74.33 | 38.50 | 65.00 |
| +RLHF-V *Yu et al.* (2023a) | 29.50 | 75.00 | 36.33 | 65.33 |
| +EOS *Yue et al.* (2024) | 31.33 | 67.00 | 38.67 | 65.67 |
| +SIMA *Wang et al.* (2024d) | 27.83 | 68.67 | 36.17 | 66.00 |
| +V-DPO *Xie et al.* (2024) | 29.50 | 72.67 | 37.83 | 67.67 |
| +Text-DPO | 31.34 | 71.67 | 37.83 | 65.67 |
| +Image-DPO | **34.17** | **75.00** | **39.33** | **68.00** |

2024e), HADPO (*Zhao et al.*, 2023c), and EOS (*Yue et al.*, 2024), by using their publicly available checkpoints. Additionally, we train models using the dataset and code provided in RLHF-V (*Yu et al.*, 2023a) and V-DPO (*Xie et al.*, 2024). All models use LLaVA-7B for a comprehensive comparisons as many paper only release 7B checkpoints. Table 5.5 shows that Image-DPO achieves the highest across all the metrics.

Besides, we evaluate the proposed Image-DPO algorithm across three VLM models—Cambrian-8B, LLaVA-1.5-7B, and LLaVA-1.5-13B—using several popular VLM benchmarks that focus on different aspects, including compositionality & biases (NaturalBench (*Li et al.*, 2024b)), general visual reasoning (MM-Vet (*Yu et al.*, 2023c)), and hallucinations (CHAIR (*Rohrbach et al.*, 2018)). The results, presented in Table 5.3, show consistent performance improvements across both datasets and models, further demonstrating the effectiveness of our Image DPO method.

**TextDPO on Corrupted Images.** Does Image-DPO's improvement stem from the objective itself, or is it merely due to training on more perturbed data? To investigate this, we conduct an ablation where we train TextDPO with corrupted images. Specifically, for each pair of $(Q, I, A_w)$ and $(Q, I, A_l)$ used in Text-DPO, we apply the same corruptions as Image-DPO to $I$ to form $(Q, I', A_w)$ and $(Q, I', A_l)$. Results show that LLaVA-v1.5-7B trained with this TextDPO variant achieves a ViLP$^{F}$-Score of 31, a ViLP$^{P}$-Score of 37.5, and 30.3 on MMVET. In contrast, ImageDPO outperforms it across all metrics, achieving a ViLP$^{F}$-Score of 34.17, a ViLP$^{P}$-Score of 39.33, and 32.3 on MMVET.

**Impact of Synthetic Image Augmentation.** When we remove the image-generation stage and fine-tune *ImageDPO* only on QIA pairs drawn from the seed datasets (COCO, VG, Text2VQA), performance slips across *every* metric: ViLP$^{\mathrm{F}}$Score declines from 34.17 to 31.17 and ViLP$^{\mathrm{P}}$Score from 39.33 to 38.00; the four novel-bias scores ($\mathrm{NB_Q}$, $\mathrm{NB_I}$, $\mathrm{NB_G}$, $\mathrm{NB_B}$) each lose roughly one point; MM-Vet drops from 32.3 to 31.4; and hallucination rates rise ($\mathrm{CHAIR^S}\downarrow$: 45 $\rightarrow$ 47, $\mathrm{CHAIR^I}\downarrow$: 12.3 $\rightarrow$ 13.6). These uniform declines confirm that synthetic images by image generative models broaden the visual distribution, inject the priors of powerful generative models, and are crucial for achieving the full gains of ImageDPO.

## 5.6 Conclusion

In conclusion, we present the ViLP benchmark to probe the challenge of visual language bias in Vision-Language Models (VLMs). By utilizing advanced image generation models and designing questions that demand visual cues for accurate responses, our benchmark includes images that defy language priors, revealing the limitations of current VLMs. Our method, Image-DPO, which incorporates self-generated VQA pairs and image corruption for training, has demonstrated promising improvements in enhancing visual reliance, as evidenced by performance gains on open-source VLMs, such as LLaVA-v1.5 and Cambrian.

# CHAPTER 6

# Towards Minimal Fine-Tuning of VLMs

*The previous chapter diagnosed shortcutting through language priors; addressing it required changing* what *the model learns. This chapter asks a complementary question: can we change* how *the model is adapted? Multimodal transformers are not monolithic—attention heads specialize, and different token spans (visual vs. textual) play distinct roles. We exploit this* ***internal model structure*** *to design a minimal, selective fine-tuning recipe that concentrates updates where they matter most for visual grounding.*

## 6.1 Introduction

Low-rank adaptation (LoRA) (*Hu et al.*, 2022a) has become a strong baseline for post-training of large language models because it exposes a small, trainable subspace while keeping the base network frozen *Schulman and Lab* (2025). However, parameter-efficient tuning (PEFT) for transformer-based vision–language models (VLMs) presents different constraints than text-only LLMs: prompts often contain long stretches of text mixed with images, and attention modules comprise many heads, even though only a subset appears to carry most of the routing signal for visual reasoning, such as grounding and pointing *Kang et al.* (2025); *Baek et al.* (2025). Existing PEFT recipes typically treat all tokens and all heads uniformly, which leaves efficiency on the table. We revisit LoRA through the lens of VLM structure and ask: what if we only adapt the parts that directly affect grounded

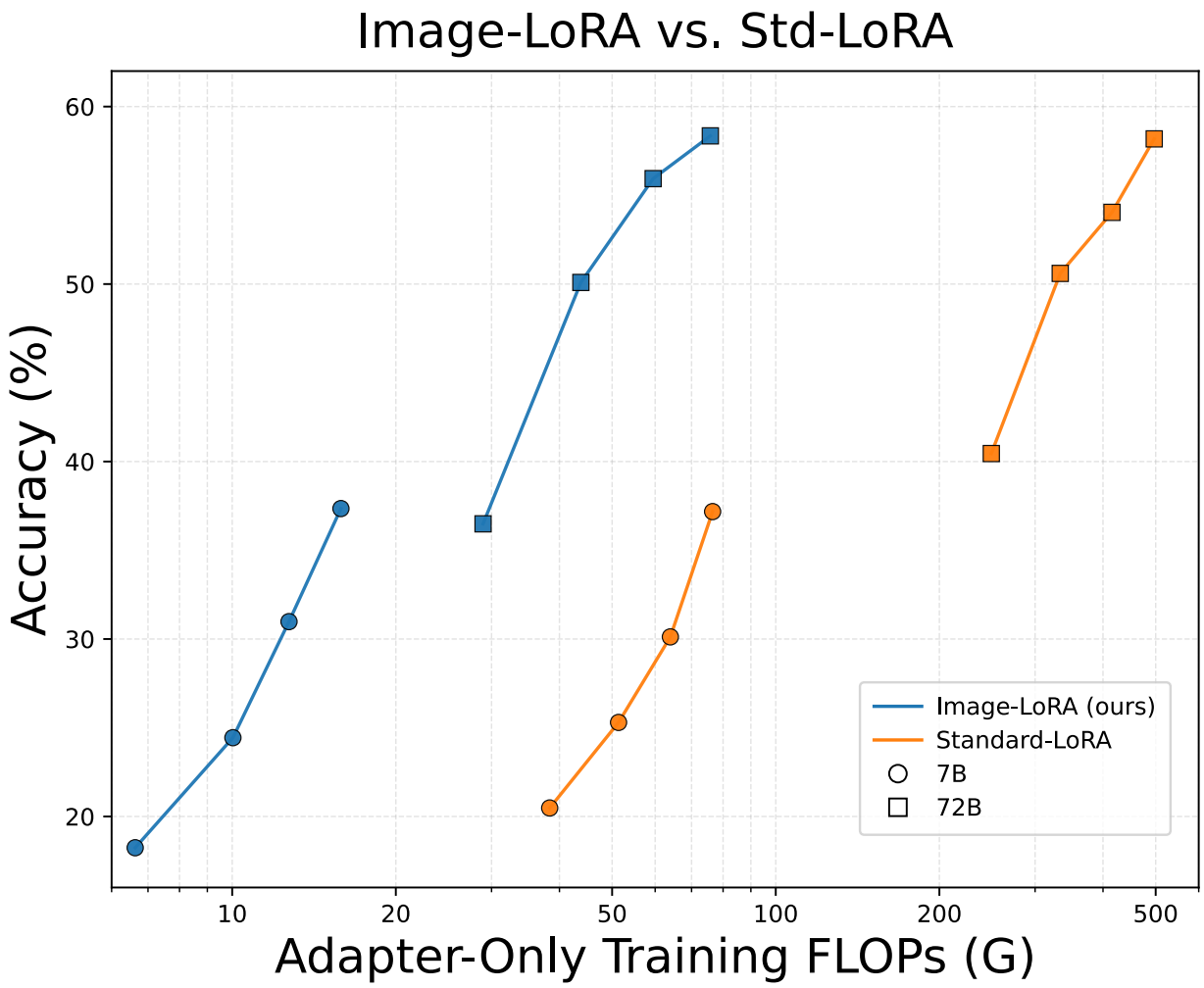



Figure 6.1: **Image-LoRA vs. Standard LoRA.** Comparison of accuracy and adapter-only training FLOPs on ScreenSpot-Pro. Points are connected across increasing input-text:image token ratios (1:2 $\rightarrow$ 1:5), where we control the ratios by dynamically adjusting image sizes. FLOPs are token-limited and computed as forward + backward multiply–adds.

vision–text reasoning?

Our first design choice is visual-token only fine-tuning. Restricting adaptation to the visual-token span reduces adapter-only training FLOPs roughly in proportion to the visual-token fraction, as it avoids computation through both the input prompt and the output text. A further benefit is that fine-tuning only on the visual-token span ensures that the adapted weights never affect the pure-text reasoning performance of the fine-tuned VLMs, as also demonstrated in our GSM8K *Cobbe et al.* (2021) experiment. This design is further supported by recent evidence that textual information

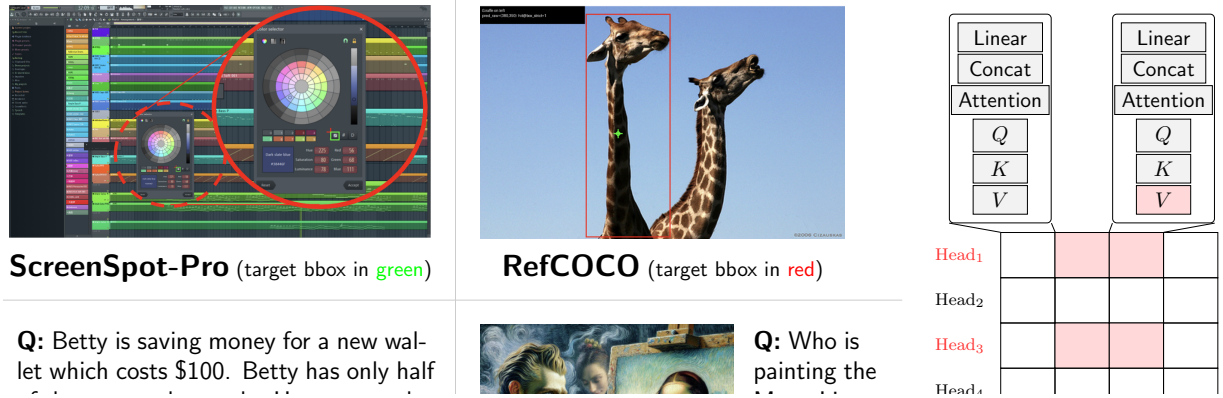



Figure 6.2: **Overview. Left:** We evaluate Image-LoRA on grounded vision–language reasoning datasets, including ScreenSpot-Pro *Li et al.* (2025b) and RefCOCO *Kazemzadeh et al.* (2014), where the model takes a text query with its system prompt and outputs a point indicating the referred object. We further evaluate on the pure text reasoning dataset GSM8K *Cobbe et al.* (2021) to confirm that Image-LoRA does not affect pure text reasoning, and on ViLP *Luo et al.* (2025a) containing VQAs both aligned with and against language priors to show that Image-LoRA may mitigate shortcutting through such priors. **Right:** Image-LoRA is applied only to selected heads along the value path of attention layers within the visual-token span (the pink area).

can be routed through visual-token channels *Wei et al.* (2025); *Cheng et al.* (2025), suggesting that adapting visual tokens alone could be powerful. To ensure that updating the attention paths on the visual-token span meaningfully influences the representations of text tokens, we compared adapting $K$ and $V$ and found that adapting only $V$ performs best. Additionally, localization-style objectives (e.g., pointing) place supervision primarily in the pixels, while the long text prompt mainly serves as task scaffolding. Concentrating adapters on the visual span allocates capacity where the loss is informative and may discourage shortcutting through language priors, as shown in our ViLP *Luo et al.* (2025a) experiment.

Second, rather than treating all attention heads equally, we ask whether a small set suffices for our tasks. A naïve approach is a finite-difference ablation: enable one head at a time, recompute the loss over the dataset, and keep the heads with the largest loss drops—but this is prohibitively expensive. To this end, we propose a head influence score estimated by attaching a rank-1 Image-LoRA to every head, running a single forward–backward pass with the task loss, and scoring each head by the squared gradient norm of its Image-LoRA parameters – a first-order empirical-Fisher proxy that identifies a high-impact subset without iterative search. After head selection, different layers may

retain different numbers of heads—unlike standard LoRA, which updates all heads per layer. To maintain stable per-layer update magnitude, we apply a selection-size layer normalization as part of Image-LoRA.

Putting these pieces together, we propose *Image-LoRA*. The adapter is injected pre-RoPE (rotary position embeddings) and pre-KV-cache so that its effect persists into decoding. Proof-of-concept experiments on screen-centric grounding and referring expression datasets (ScreenSpot-Pro *Li et al.* (2025b) and RefCOCO *Kazemzadeh et al.* (2014)) span a wide range of input-text:image token ratios (1:1/2, 1:1, 1:2, 1:3, 1:4, 1:5) along with QWen2.5-VL-7B, QWen2.5-VL-72B, and LLaVA-NeXT-7B models. We enforce the chosen ratio during both training and evaluation to probe regimes from text-heavy to image-heavy. Our results, shown in Figure 6.1, demonstrate that Image-LoRA retains accuracy while reducing trainable parameters and adapater-only training FLOPs, especially in image-heavy settings. Furthermore, experiments on GSM8K *Cobbe et al.* (2021) show that Image-LoRA preserves pure-text reasoning performance, whereas standard LoRA may not. We also evaluate on ViLP *Luo et al.* (2025a), a VQA dataset deliberately constructed to follow or break language priors, where Image-LoRA outperforms standard LoRA. In the ablation studies, we further evaluate the proposed head-selection method against correlation-based, global-random, and per-layer-random baselines, as well as examine various design choices for Image-LoRA.

Although our proof-of-concept experiments above show some promising signals for Image-LoRA, our experimental scale is limited, and fundamental constraints inherent to the Image-LoRA formulation remain and require further investigation, as discussed in Section 6.5.

Collectively, we present a vision-orientated PEFT recipe for transformer-based VLMs. By restricting adaptation to the visual-token span, stabilizing updates when adapting a subset of attention heads, and preserving pure-text reasoning behavior, Image-LoRA offers a structured step toward enabling web-scale vision pre-training on top of existing reasoning LLMs (*Jaech et al.*, 2024).

## 6.2 Related Works

**PEFT of VLMs.** A common approach to efficient fine-tuning of VLMs is to reduce the number of parameters to update, which is often called parameter-efficient fine-tuning (PEFT). One important milestone in PEFT is leveraging the reparameterization of the parameters of models, often notably represented by low-rank adaptation (LoRA) (*Hu et al.*, 2022a), which has also been shown to be effective for VLMs (*Sung et al.*, 2022; *Gao et al.*, 2023; *Zong et al.*, 2024; *Zanella and Ben Ayed*, 2024). Recently, Schulman et al. *Schulman and Lab* (2025) examined the effects of applying LoRA either only to attention layers or to all layers, including MLP/MoE layers, a topic explored in prior works (*Zaken et al.*, 2021; *Zhao et al.*, 2020; *Sung et al.*, 2021; *Guo et al.*, 2020; *Fu et al.*, 2023b; *Wang et al.*, 2025). In contrast, our work explores a more structured approach—applying LoRA selectively to attention heads and to the visual token span only. Meanwhile, *Shuttleworth et al.* (2024) reported findings that contradict *Schulman and Lab* (2025), showing that LoRA can behave differently from full fine-tuning, with both studies focusing on pure text-based benchmarks. In contrast, we focus on visual reasoning tasks, including both grounding and VQA tasks.

**Token selection and reduction for VLMs.** Token selection and reduction have been applied to improve VLM inference efficiency. Token pruning removes unimportant input tokens based on relevance criteria. FastV (*Chen et al.*, 2024b) ranks visual tokens by attention scores, while Sparse-VLM (*Zhang et al.*, 2024b) first filters relevant text tokens and then keeps the most attended visual tokens. Token merging instead combines similar tokens to reduce sequence length. PuMer (*Cao et al.*, 2023) prunes low-saliency visual tokens and merges similar tokens within the same modality, and VisionZip (*Yang et al.*, 2025b) merges non-dominant visual tokens based on attention scores. On the other hand, we explore a different angle through token selection—investigating how constraining the set of input tokens can make the fine-tuning of VLMs (rather than inference) more efficient by applying Image-LoRA to the visual-token span only. This offers two benefits: (1) it saves FLOPs from both input and output text, and (2) such LoRA fine-tuning does not alter the pure-text reasoning behavior of VLMs.

**Attention head selection for VLMs.** Prior work shows that attention heads in LLMs and VLMs differ in their task importance. *Michel et al.* (2019) demonstrate redundancy among heads in language models and prune them by measuring loss sensitivity to head masking, while *Voita et al.* (2019); *Zhang et al.* (2023b) similarly assess head importance via backpropagated gradients and achieve comparable performance after pruning. Furthermore, *Khaki et al.* (2025) accelerate LLM fine-tuning with an SVD-based sparsity estimator over the model's pre-trained weight matrix. *Kang et al.* (2025) identify "localization heads" crucial for visual grounding and leverage their attention maps at inference, and *Zhang et al.* (2025) enhance perception by using the most attentive layer for test-time visual cropping. This paper develops an efficient first-order approximation for head selection as a simple complement to our proposed Image-LoRA, and shows that, when combined with the proposed layer-wise head-selection size normalization, our method performs well even with irregularly selected attention heads.

## 6.3 Method

We propose a parameter-efficient fine-tuning recipe for steering transformer-based VLMs on visual reasoning tasks while minimizing trainable parameters and adapter-only FLOPs, while preserving the pure-text reasoning ability of the VLM. We focus on tasks where visual reasoning is the dominant component (e.g., text-conditioned visual localization) and leave extensions to other task types for future work.

Our approach achieves *minimal* adaptation through three forms of selectivity: 1. **Token selectivity (visual-only):** We adapt representations only for visual tokens, rather than for all tokens (including text). This contrasts with standard PEFT approaches (e.g., LoRA) that introduce adapters in projections shared by every token. 2. **Head selectivity (subset of attention heads):** We update only a subset of attention heads and introduce a selection procedure to identify a diverse, salient set of heads relevant to visual reasoning. 3. **Projection selectivity (value-only):** In contrast to LoRA, which typically inserts low-rank adapters into the query ($Q$), key ($K$), value ($V$), and output ($O$) projections, we attach adapters only to the value ($V$) projections of the selected heads and only for

visual tokens.

### 6.3.1 Background

**Transformer Multi-Head Attention.** For a sequence of length $T$, let the input and output of a transformer layer be $X, Y \in \mathbb{R}^{T \times d_{\text{hidden}}}$, where $d_{\text{hidden}}$ is the model width. Multi-head attention can be written as

$$Y = \text{MultiHead}(X) = \text{Concat}(O^{(1)}, \ldots, O^{(H)})\, W_O$$

$$\text{where } O^{(h)} = \text{softmax}\left(\frac{Q^{(h)} K^{(h)\top}}{\sqrt{d_k}} + Mask\right) V^{(h)},$$

$$Q^{(h)} = XW_Q^{(h)},\ K^{(h)} = XW_K^{(h)},\ V^{(h)} = XW_V^{(h)}.$$

Here $H$ is the number of attention heads. $Mask \in \mathbb{R}^{T \times T}$ is the causal mask with $Mask_{ij} = 0$ if $j \leq i$ and $Mask_{ij} = -\infty$ otherwise; the softmax is applied row-wise. The query and key dimension and value dimension are denoted by $d_k$ and $d_v$, respectively (for simplicity, we assume $d_k = d_v = d_{\text{hidden}}/H = d_{\text{head}}$), and the learnable projection matrices are $W_{\{Q,K,V\}}^{(h)} \in \mathbb{R}^{d_{\text{hidden}} \times d_{\text{head}}}, W_O \in \mathbb{R}^{(H \times d_{\text{head}}) \times d_{\text{hidden}}}$.

**LoRA.** LoRA is a parameter-efficient fine-tuning method that models updates to weight matrices as low-rank adaptations. LoRA represents a linear map such as $V = XW_V$ (superscript $(h)$ avoided for brevity) as a low-rank decomposition with $r \ll \min(d_{\text{head}}, d_{\text{hidden}})$,

$$\Delta W_V = AB, \quad A \in \mathbb{R}^{d_{\text{hidden}} \times r},\ B \in \mathbb{R}^{r \times d_{\text{head}}},$$

yielding the modified forward pass

$$V = X(W_V + \Delta W_V) = XW_V + XAB \cdot s, \qquad s = \frac{\alpha}{r}\gamma,$$

where $\alpha$ is a scaling hyperparameter, $\gamma$ is a learned scalar gate, and $r$ is the adaptation rank. The original weight $W_V$ is frozen; only $(A, B, \gamma)$ are trained.

### 6.3.2 Image-LoRA

**Vision Tokens Only.** The first distinguishing property of our method is that we only learn to update representations corresponding to vision tokens, unlike the standard LoRA which treats both vision and text tokens equally. Concretely, let the input prompt contain $T$ tokens, of which $T_v = |\mathscr{I}_v|$ correspond to vision tokens, where $\mathscr{I}_v \subseteq \{1,\ldots,T\}$ denotes the indices of vision tokens. We present the learning approach for adapting the value token representations $V^{(h)}$. Later in the section we justify this choice and consider alternatives. For a given attention head, we learn to update the vision token representations only

$$\tilde{v}_j^{(h)} = v_j^{(h)} + \Delta v_j^{(h)}; \quad \text{where } \Delta v_j^{(h)} := 0 \text{ for } j \notin \mathscr{I}_v$$

and each $\Delta v_j^{(h)}$ is learned via LoRA as described next.

**Selected Heads Only.** Second, we apply updates to only a subset of attention heads rather than all heads. This provides finer control—adapting only those heads most relevant to the task while leaving others unchanged. Let $\mathscr{H}^{\text{chosen}} \subseteq \{1,\ldots,H\}$ be the set of selected heads (see Section 6.3.3 for specific head selection criteria) and $N^{\text{chosen}} = |\mathscr{H}^{\text{chosen}}|$.

Combining the two ideas above, we propose to learn parameters $\Delta W_V^{(h)}$ such that

$$\tilde{V}^{(h)} = XW_V^{(h)} + \gamma_h M X \Delta W_V^{(h)},$$

where $\gamma_h = \mathbb{I}[h \in \mathscr{H}^{\text{chosen}}]$ and $M \in \{0,1\}^{T\times T}$ is a diagonal matrix with $M_{i,i} = \mathbb{I}[i \in \mathscr{I}_v]$. This matrix formulation is provided for clarity; in practice, we parameterize $\Delta W_V^{(h)}$ using a LoRA decomposition:

$$\Delta W_V^{(h)} = AB^{(h)}s, \qquad s = \tfrac{\alpha}{r}\,\gamma\nu(N^{\text{chosen}}),$$

where $A \in \mathbb{R}^{d_{\text{hidden}}\times r}$ and $B^{(h)} \in \mathbb{R}^{r\times d_{\text{head}}}$ are learnable matrices, and the scaling factor $s$ includes the normalization term $\nu(N^{\text{chosen}})$, which we describe next. In our implementation, $A$ is shared across

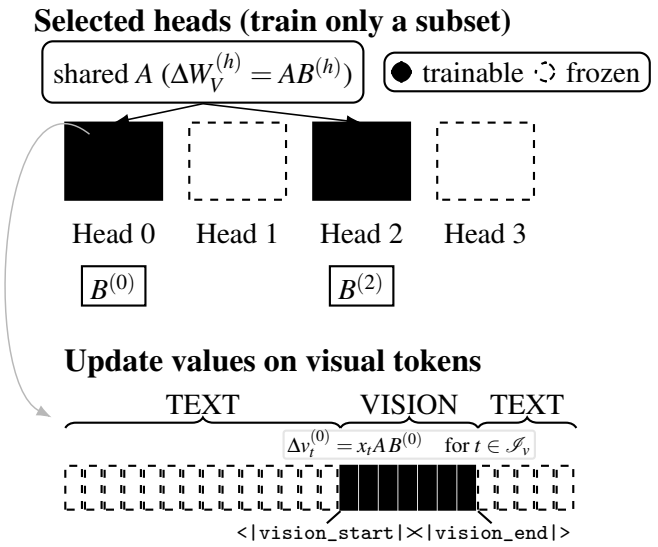



Figure 6.3: **Top**: Share one $A$ per layer across its selected heads; learn $B^{(h)}$ only for the selected heads. **Bottom**: For the selected heads, we update the *value* vectors $v_t$ of the attention layer only on the visual-token span $\mathscr{I}_v$ and not on the text-token positions.

heads within a layer, while $B^{(h)}$ is head-specific, motivated by two factors: (1) since $d_{\text{hidden}} \gg d_{\text{head}}$[1], sharing $A$ across a layer greatly reduces parameters and training FLOPs (detailed numbers for both 7B and 72B models in our ablation studies), and (2) head-specific parameters are required for the head-selection algorithm to compute per-head influence scores.

**Head-selection Normalization.** The standard LoRA activates *all* heads per layer, so the per-layer update magnitude is comparable across layers. With head selection, the number of chosen heads in a layer varies. To prevent the summed increment from growing with the number of chosen heads, we additionally apply a *selection size normalization*:

$$\nu(N^{\text{chosen}}) = \frac{1}{\sqrt{N^{\text{chosen}}}}.$$

This design follows from a variance-preserving argument that ensures the variance of $\sum_{h \in \mathscr{H}^{\text{chosen}}} \Delta V^{(h)}$ remains invariant with respect to the number of chosen heads, assuming uncorrelated heads. See Section E.1.2 of the Appendix for more details.

[1]For example, in Qwen2.5-VL-7B, $d_{\text{hidden}} = 3584$, $d_{\text{head}} = 128$.

**Projection Selectivity.** The standard LoRA can be applied to any of the query $Q^{(h)}$, key $K^{(h)}$, value $V^{(h)}$ or output $O^{(h)}$ representations. If we restrict the analysis to a single layer, updating the $Q^{(h)}$ or $O^{(h)}$ representations of vision tokens does not influence the output representations of text tokens. This is because

$$o_i^{(h)} = \text{softmax}\left(\frac{q_i^{(h)} K^{(h)\top}}{\sqrt{d_k}} + Mask_i\right) V^{(h)},$$

and thus $o_i^{(h)}$ depends only on the query for the token at $i$, together with the shared keys $K^{(h)}$ and values $V^{(h)}$. Consequently, modifying $q_j^{(h)}$ or $o_j^{(h)}$ for any vision token at $j \in \mathscr{I}_v$ has no effect on the output $o_i^{(h)}$ of any text token at $i \notin \mathscr{I}_v$. On the other hand, $V^{(h)}$ and $K^{(h)}$ determine 'what to look' and 'where to look' in the image respectively, from the perspective of text tokens, and updating $V^{(h)}$ and $K^{(h)}$ solely on the vision tokens is more aligned with our goals. Based on our ablations, we find that updating the value representation $V^{(h)}$ of vision tokens yields the optimal performance and outperforms alternatives such as updating only $K^{(h)}$ or both $K^{(h)}$ and $V^{(h)}$.

**Summary of Differences.** Compared to the standard LoRA, Image-LoRA (i) applies updates only to visual tokens, reducing adapter-only training FLOPs by skipping computation over both the input prompt text and output answer text, (ii) adapts only a subset of selected heads, with selection-size normalization to stabilize the magnitude of layer-wise updates, and (iii) shares $A$ while keeping $B^{(h)}$ head-specific, so the parameter count scales with $|\mathscr{H}^{\text{chosen}}|$ instead of $H$, and the head-specific $B$ enables head selection. These design choices yield substantial parameter and adapter-only training FLOP reductions while effectively modulating value representations. Relative to the standard LoRA, our approach reduces the FLOPs by a factor of $O(T_v/T \times |\mathscr{H}^{\text{chosen}}|/H)$.

### 6.3.3 Head Selection

**First-order Head Importance Estimation.** Our goal is to identify the attention heads $h$ in the model that would most effectively reduce the training loss when their value matrices $V^{(h)}$ are updated. A naïve way to rank heads is to modify one head at a time, recompute the loss for all samples, and select the heads that yield the largest decrease. However, this finite-difference

procedure is prohibitively expensive. Instead, we adopt a first-order Taylor approximation that scores *all* heads in one backward pass. For parameters $\theta$ and loss $\ell$, the first-order Taylor expansion gives

$$\Delta\ell = \ell(\theta + \delta\theta) - \ell(\theta) \approx \langle \nabla_\theta \ell, \delta\theta \rangle,$$

where higher-order terms $O(\|\delta\theta\|^2)$ are neglected. Choosing the steepest direction of descent $\delta\theta = -\nabla_\theta \ell$ leads to a change is loss as

$$\Delta\ell \approx -\|\nabla_\theta \ell\|_F^2$$

Thus, $\|\nabla_\theta \ell\|_F^2$ can be used as a measure of the sensitivity of the loss to parameters $\theta$.

**Head Importance.** To measure the importance of each head, we adopt a setup analogous to the Image-LoRA factorization described in §6.3.2. Following standard practice, we initialize $A$ with Gaussian noise and set $B^{(h)}$ to zeros, with $\gamma = 1$ and LoRA rank $r = 1$ for efficiency. The importance of head $h$ is then defined as

$$I(h) = \|\nabla_{B^{(h)}} \ell\|_F^2$$

which we refer to as the *influence score*. Since $A$ is shared and frozen while $B^{(h)}$ is head-specific and trainable, the gradient magnitude on $B^{(h)}$ directly reflects how influential that head is.

**Head Diversity.** Besides importance, we consider a diversity factor, which we found beneficial in our experiments:

$$F(h) = (\nabla_{B^{(h)}} \ell) \odot (\nabla_{B^{(h)}} \ell),$$

where $\odot$ denotes the Hadamard (element-wise) product.

**Head Selection Procedure.** We next select a subset of heads that balances importance and diversity. To determine the number of heads to select from each layer, we first compute the aggregate

importance of layer $L$ and convert it into a normalized weighting that guides head allocation:

$$\Phi_L = \sum_{h \in L} I(h), \qquad p_L = \frac{(\Phi_L)^\tau}{\sum_{L'} (\Phi_{L'})^\tau}$$

where $\tau \geq 0$ is a temperature parameter controlling how strongly importance is concentrated across layers. Given a total budget of $K_{\text{sel}}$ heads, we allocate a per-layer budget using $p_L$ and then perform a simple selection within each layer with considering diversity $F(h)$; See Appendix E.1.1 and Algorithm 3 for the full procedure.

## 6.4 Experiments

Our experiments test whether *Image-LoRA* can match the capabilities of the *standard LoRA* while using fewer trainable parameters and lower adapter-only FLOPs evaluated on grounded vision-text reasoning tasks. Particularly, we examine QWEN2.5-VL-7B & 72B and LLaVA-NeXT-7B on ScreenSpot-Pro *Li et al.* (2025b), RefCOCO *Kazemzadeh et al.* (2014), GSM8K *Cobbe et al.* (2021), and ViLP (*Luo et al.*, 2025a) datasets. Then, we conduct extensive ablation studies on the design choices for Image-LoRA to demonstrate the effectiveness of our head-selection algorithm, the need of layer-wise normalization, the inferior performance of adapting both K and V compared to V alone, and other design variants.

### 6.4.1 Image-LoRA vs. Standard LoRA

We evaluate *Image-LoRA* and *standard LoRA* sweeping from text-heavy to image-heavy settings. We primarily study QWEN2.5-VL-7B and QWEN2.5-VL-72B on ScreenSpot-Pro and RefCOCO, whose vision encoders allow accurate control of the image token numbers to yield different sceneries. To verify that Image-LoRA does not harm pure text reasoning, we also report empirical results on GSM8K, where the standard LoRA degrades performance while our Image-LoRA preserves the base model's text reasoning capabilities. Moreover, evaluations on ViLP indicate that restricting adaptation to visual tokens might reduce shortcutting through language priors. Finally, we further

Table 6.1: **Main results across input text:image ratios.** FLOPs (G) denotes the total adapter-only training-time floating-point operations, measured in billions ($10^{-9}$). The average fine-tuned parameter count for Image-LoRA for 7B and 72B models are, respectively, 629,781 and 3,011,629. The respective parameter counts for Std-LoRA are 2,523,192 and 16,384,160, respectively.

| | | QWEN2.5-VL-7B | | | | | | QWEN2.5-VL-72B | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | SCREENSPOT-PRO | | | | REFCOCO | | SCREENSPOT-PRO | | | | REFCOCO | |
| Text:Img Token Ratio | | 1:5 | 1:4 | 1:3 | 1:2 | 1:1 | 1:1/2 | 1:5 | 1:4 | 1:3 | 1:2 | 1:1 | 1:1/2 |
| Avg #Image Tokens | | 4,195 | 3,367 | 2,534 | 1,677 | 851 | 426 | 4,195 | 3,367 | 2,534 | 1,677 | 851 | 426 |
| Acc. (%) | Base | 22.38 | 14.97 | 8.95 | 9.12 | 90.80 | 83.00 | 46.47 | 45.44 | 43.89 | 32.19 | 93.60 | 92.60 |
| | Std-LoRA | 37.18 | 30.12 | **25.30** | **20.48** | 93.40 | **93.20** | 58.18 | 54.04 | **50.60** | **40.45** | 95.20 | **95.20** |
| | Image-LoRA | **37.35** | **30.98** | 24.44 | 18.24 | **93.60** | 93.00 | **58.35** | **55.94** | 50.09 | 36.49 | **95.80** | 94.80 |
| FLOPs (G) | Std-LoRA | 76.53 | 63.99 | 51.38 | 38.38 | 25.86 | 19.42 | 496.9 | 415.5 | 333.7 | 249.2 | 167.9 | 12.61 |
| | Image-LoRA | **15.85** | **12.72** | **10.03** | **6.63** | **3.36** | **1.61** | **75.80** | **59.49** | **43.80** | **28.94** | **15.05** | **7.54** |

test our approach on LLAVA-NEXT-7B.

**Controlling the input-text:image token ratio.** We control the ratio between number of input text tokens and number of image tokens by varying the image resolution while keeping the chat template fixed. As Qwen models tokenize images on a regular patch grid, the number of visual tokens, $T_v$, grows monotonically with the processed area. The text side effectively contains about $T_{\text{text}} \approx 850$ tokens, mostly from system prompts, while each per-sample query remains short. The whole process only resamples the image while preserving its original aspect ratio as much as possible. See Section E.2.1 for more details.

**Task and Benchmarks.** For our main experiments, we consider a grounding task with a single-line target `point_2d:[x, y]`; we optimize cross-entropy over this string. We report **Accuracy** as the fraction of predicted $(x, y)$ that fall inside the ground-truth box in the model's effective pixel space, and also report trainable parameters and adapter-only training FLOPs.

- **ScreenSpot-Pro *Li et al.* (2025b) (UI grounding):** high-resolution interface screenshots paired with natural-language instructions and click/box annotations. Because screenshots are often $> 3000 \times 2000$, we use *image-heavy* input text:image ratios of 1:2, 1:3, 1:4, and 1:5. Average image token counts are in Table 6.1; example image resolutions per setting are in Table E.1. We use a fixed random split: 1000 examples for *training + head selection* and the remaining 581 for *evaluation*.

- **RefCOCO *Kazemzadeh et al.* (2014) (referring expressions):** natural descriptions of target objects in COCO images with bounding boxes (and segmentations for a subset). As COCO images are relatively small ($\approx 600\times640$), we use ratios of 1:1 and 1:1/2 (input text:image). We sample 2000 `val` examples for *training* + *head selection* and evaluate on 500 examples from `test` with no overlap.

**Training.** We compare our approach against the standard LoRA approach under different configurations.

- **Image-LoRA** adapts only the $V$ projection in attention layers and only on visual-token spans, using rank $r$=8 and scale $\alpha$=16 without losing generality. For each setting (i.e., different text:image ratio), we apply the proposed head selection strategy, choosing 28 out of 112 heads for QWEN2.5-VL-7B and 80 out of 640 for QWEN2.5-VL-72B. Note that 28 and 80 correspond to the number of layers in these models, respectively.
- The **standard LoRA (Std-LoRA)** baseline adapts $Q$ and $V$ on all attention heads and all tokens with the same $r$ and $\alpha$. We choose $Q,V$ because this is the most widely used configuration and is also the default setting in the official documentation

. We also compare with the $V$-only Std-LoRA in terms of both FLOPs and performance, and further analyze other key configurations of standard LoRA in our ablation studies.

**Results.** Table 6.1 groups settings by dataset and ratio, and for each setting compares *Base (frozen)*, *Std-LoRA (QV, all tokens, all heads)*, and *Image-LoRA (V, visual tokens only, 28 heads)* in terms of accuracy, adapter FLOPs, and trainable parameters. Notably, the FLOPs of Image-LoRA are much lower than those of the standard LoRA, due to two factors: (1) using fewer heads via the proposed head selection algorithm, which reduces trainable parameters; and (2) computing only over the visual token span, excluding input prompts and output answers, which saves computation. This also yields the expected scaling behavior: Image-LoRA FLOPs are proportional to the image-token count (so $1:4 \approx 2\times1:2$ and $\approx 4\times1:1$), while Std-LoRA counts all tokens and thus scales with the

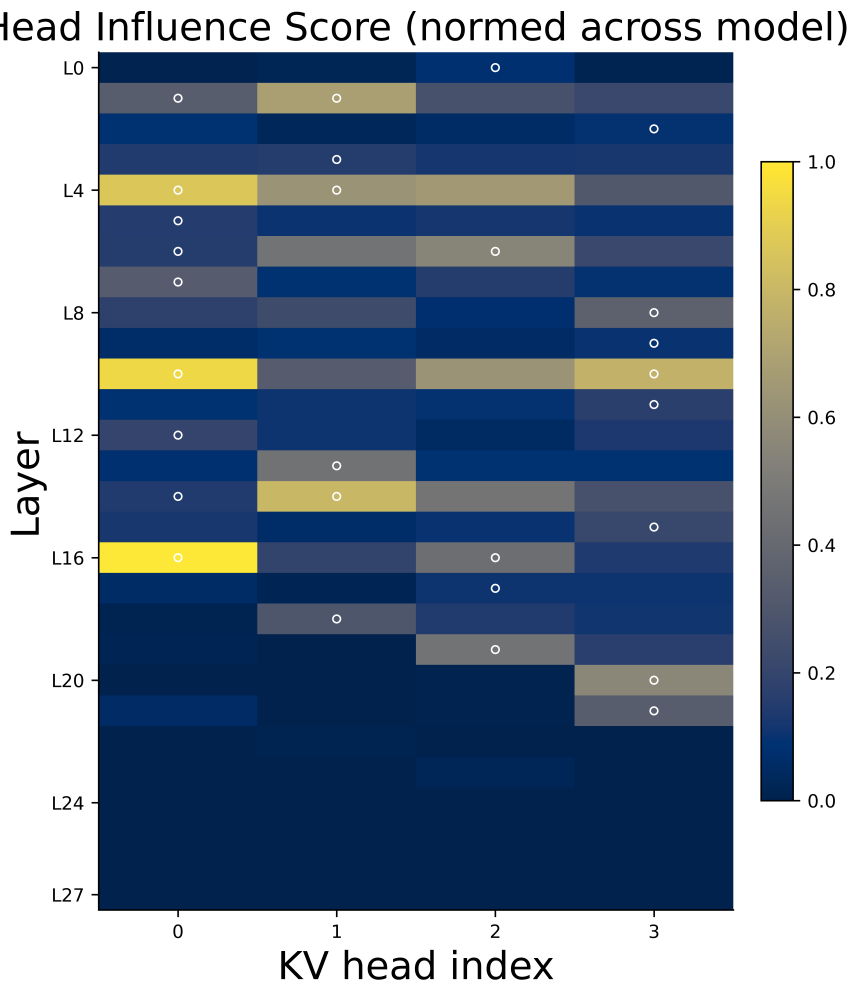



Figure 6.4: Head selection for Qwen2.5-VL-7B under a input-text:image token ratio of 1:2.

overall sequence length (prompt text + image + answer text). As a result, restricting adaptation to image tokens (i.e., our Image-LoRA) is particularly beneficial in settings with long textual contexts.

Since choosing $QV$ for Std-LoRA adds FLOPs relative to $V$-only configuration, we also compare a **$V$-only Std-LoRA vs. our Image-LoRA**. On ScreenSpot-Pro, Image-LoRA reduces adapter-only FLOPs by 43.0%, 45.3%, 48.7%, and 54.6% at text:image ratios 1:5, 1:4, 1:3, and 1:2, respectively. Averaged over these settings, Image-LoRA yields a $\approx 48\%$ FLOPs reduction (Detailed numbers are provided in Appendix E.2.2). These gains come from Image-LoRA's design—fewer adapted heads with the proposed head selection algorithm and computation restricted to the visual-token span. Regarding performance, $V$-only Std-LoRA achieved consistent worse results than $QV$ Std-LoRA, as shown in Table 6.4.

In terms of accuracy, Image-LoRA achieves comparable performance in image-heavy settings such as ScreenSpot-Pro (1:5, 1:4) and RefCOCO (1:1), while performing worse in those image-token less scenarios. The trainable parameters of Image-LoRA are approximately one-quarter those of Std-LoRA for 7B models and one-fifth for 72B models.

We show one example of our head selection for QWEN2.5-VL-7B in Figure 6.4, where the selected heads are distributed from the early layers to the middle-late layers. Similar patterns are observed in QWEN2.5-VL-72B. Due to space limitations, additional visualizations, analyses, and ablation stduies are provided in Appendix E.2.1.

**Does Image-LoRA Affect Text Only Reasoning?** Because Image-LoRA applies adaptive weights only on the visual-token span, it does not interfere with language ability on pure-text inputs fundamentally. To show this empirically, we evaluate on the GSM8K *Cobbe et al.* (2021) test set—a collection of 1,319 grade-school math word problems all in text form (no images). Qwen2.5-VL base attains 25.55%, and our Image-LoRA fine-tuned achieves the same 25.55%. In contrast, a standard LoRA baseline fine-tuned on the same GSM8K data and text:image ratio drops to 24.79%. This result supports that Image-LoRA preserves pure-text reasoning while the standard LoRA, which updates all tokens, may degrade it.

**Can Image-LoRA Mitigage Overfitting to Language Priors?** Because Image-LoRA updates only the visual-token span, it might be less susceptible to overfitting language priors than Std-LoRA, which adapts parameters on all tokens. We evaluate this on ViLP (*Luo et al.*, 2025a), where each question is paired with three images: one whose answer can be inferred from the question text alone ("follow") and two whose correct answers contradict the text prior and therefore require visual evidence ("against"). For example, for the question "Who is painting the Mona Lisa in the image?", the text prior suggests *Leonardo da Vinci*, whereas the correct answer may be *Vincent van Gogh* depending on the image (Figure 6.2). Using the same backbone and hyperparameters as in the main experiments, we fine-tune on the "follow" split and evaluate on the corresponding "against" images (i.e., same questions, out-of-distribution answers). Std-LoRA attains 5.00% accuracy, whereas Image-LoRA reaches 12.67%. The results suggest that constraining adaptation to visual tokens may

help mitigate shortcutting via language priors.

**Applicability to Other VLMs.** We also evaluate LLaVA-NeXT-7B, comparing Image-LoRA with Std-LoRA. Because LLaVA-NeXT uses a quantized ViT grid with discrete high-resolution tiling, the image-token count changes only at tiling thresholds, which prevents precise control of the text:image token ratio; therefore we keep the original image resolution and match all other settings to the Qwen experiments. On ScreenSpot-Pro, accuracy remains near-zero before and after fine-tuning for both methods, so we report only RefCOCO. At the original resolution, the average token counts are $T_{\text{text}}$=133.4 and $T_v$=2276.3. The LLaVA-v1.6-Mistral-7B base achieves 13% on the RefCOCO test set; Std-LoRA reaches 91.2% and Image-LoRA 92.0%. The corresponding trainable parameters are 3,407,936 (Std-LoRA) vs. 786,454 (Image-LoRA), and training FLOPs are 49.49 and 10.74 G, respectively. These results indicate that Image-LoRA remains competitive for LLaVA-NeXT architecture.

### 6.4.2 Ablations on Image-LoRA

We examine Image-LoRA designs in terms of (1) head-selection strategy, (2) how the head budget affects performance, (3) whether further adapting $K$ on top of $V$ brings additional benefits, (4) the types of layer-wise normalization, and (5) how parameters and FLOPs change when we not share A cross a layer. All experiments are conducted on the Qwen2.5-VL-7B model with rank $r$=8, LoRA scale $\alpha$=16, ScreenSpot-Pro dataset, and the same training hyperparameters as in the main experiments.

**Head Selection Strategy.** We compare our head selection strategy against the following baselines – *Global-Rand*: randomly sample 28 heads from the full model, *PerLayer-Rand*: sample exactly one head per layer (a total 28 heads), and a correlation-map heuristic (*CorrMap*) inspired by grounding saliency: we compute per-head correlation maps by comparing each attention head's image-token attention to the ground-truth region across samples, derive a per-head alignment score from these maps, and select the 28 heads with the highest scores for later use (details in Appendix E.2.2). The results are shown in Table 6.2, demonstrating that our proposed approach consistently achieves the

Table 6.2: **Head selection methods ablation on Qwen2.5-VL-7B.** All methods adapt $K_{\text{sel}}$=28 heads. Entries are accuracy (%).

| Method | 1:5 | 1:4 | 1:3 | 1:2 |
|---|---|---|---|---|
| Global-Rand | 34.42 | 28.40 | 21.86 | 16.87 |
| PerLayer-Rand | 34.08 | 27.78 | 20.31 | 17.90 |
| CorrMap | 32.87 | 25.99 | 20.14 | 16.18 |
| Ours | **37.35** | **30.98** | **24.44** | **18.24** |

Table 6.3: **KV-budget ablation on Qwen2.5-VL-7B.** We vary $K_{\text{sel}} \in \{1, 7, 28, 112\}$. Entries are accuracy (%).

| Method | 1:5 | 1:4 | 1:3 | 1:2 |
|---|---|---|---|---|
| Base (frozen) | 22.38 | 14.97 | 8.95 | 9.12 |
| 1 head | 28.23 | 16.52 | 12.05 | 10.50 |
| 7 heads | 30.81 | 24.27 | 18.42 | 15.15 |
| 28 heads (ours) | **37.35** | **30.98** | 24.44 | 18.24 |
| 112 heads | 35.11 | 28.57 | **25.82** | **19.28** |

highest scores across all the ratios.

**Number of Chosen Heads.** We vary the KV-head budget while keeping the selection method fixed to our one-shot influence head-selection algorithm. Qwen2.5-VL-7B has 28 layers and 112 KV heads in total, and we sweep among $\{1, 7, 28, 112\}$ heads. As shown in Table 6.3, accuracy generally increases with more heads; however, more is not always better—at ratios 1:5 and 1:4, selecting 28 heads outperforms using all 112 heads, highlighting the effectiveness of our proposed selection algorithm.

**Updating $K$ in addition to $V$.** Following the discussion in Section 6.3.2-Projection Selectivity, we test adding LoRA on the $K$ path in addition to the $V$ path on visual tokens, where we observe performance degradation compared to the $V$-only, as shown in the upper half of Table 6.4 for both $K$ only and $V + K$ configurations.

**Different keys for Std-LoRA.** For Std-LoRA, we further explore the $V$-only variant and $QKVO$ variant on top of the $QV$ configuration studied previously. As shown in the lower half of Table 6.4, Std-LoRA with $QV$ achieves the best performance across most settings. We list more information, such as trainable parameters, in Appendix E.2.2.

Table 6.4: **Updating $K$ in ImageLoRA + different LoRA keys for Std-LoRA.** Entries are accuracy (%). Image-LoRA with $V$ only is our proposed method.

| Method | Adapters | 1:5 | 1:4 | 1:3 | 1:2 |
|---|---|---|---|---|---|
| Image-LoRA | $K$ only | 29.60 | 24.10 | 19.10 | 13.77 |
| Image-LoRA | $V+K$ | 34.77 | 25.65 | 22.20 | 15.49 |
| Image-LoRA | $V$ only | **37.35** | **30.98** | 24.44 | 18.24 |
| Std-LoRA | $QKVO$ | 34.77 | 28.74 | **25.30** | **21.51** |
| Std-LoRA | $V$ | 35.28 | 29.60 | 24.78 | 19.45 |
| Std-LoRA | $QV$ | 37.19 | 30.12 | **25.30** | 20.48 |

Table 6.5: **Effect of selection-size normalization on Qwen2.5-VL-7B.** All runs use the proposed head selection procedure, $K_{\text{sel}}$=28, $V$-only, $r$=8. Entries are accuracy (%).

| Normalization | 1:5 | 1:4 | 1:3 | 1:2 |
|---|---|---|---|---|
| None ($\nu(S)$=1) | 34.60 | 29.43 | 23.41 | 17.73 |
| Linear ($\nu(S)$=$1/S$) | 35.97 | 27.71 | 24.44 | 17.21 |
| Ours ($\nu(S)$=$1/\sqrt{S}$) | **37.35** | **30.98** | **24.44** | **18.24** |

**Selection-size Normalization.** With head selection, the number of adapted heads per layer $N^{\text{chosen}}$ varies, so the raw LoRA increment $\sum_{h\in\mathscr{H}^{\text{chosen}}} \Delta V^{(h)}$ can grow with $N^{\text{chosen}}$. We therefore scale the per-layer update as $s = \frac{\alpha}{r}\gamma\nu\big(N^{\text{chosen}}\big)$, and compare two choices for $\nu(\cdot)$: **None** ($\nu(S)$=1), which lets the expected update energy grow $\propto N^{\text{chosen}}$, and **Linear** ($\nu(S)$=$1/S$), which keeps the summed increment roughly constant but shrinks expected energy as $\propto 1/N^{\text{chosen}}$. None-normalization tends to over-scale layers where many heads are selected, while Linear-normalization may over-dampen layers with numerous selected heads. Both typically yield lower accuracy than our proposed $\nu(S)$=$1/\sqrt{S}$, as verified in Table 6.5.

**Effect of sharing $A$.** Sharing $A$ across heads within each layer reduces adapter-only training FLOPs and parameters on ScreenSpot-Pro across a range of text:image token ratios. With Qwen2.5-VL-7B, the average adapter-only FLOPs drop by $\approx 21.1\%$ and the parameter count by a similar fraction ($\approx$ 172,032 fewer). With Qwen2.5-VL-72B, the average FLOPs reduction is $\approx 26.7\%$ with a comparable parameter decrease of $\approx 26.6\%$ ($\approx$ 1,064,960 fewer).

## 6.5 Limitations and Discussions

The proposed Image-LoRA introduces fundamental challenges for batching, as image-token and text-token lengths vary across samples and must be handled efficiently. Both training and inference further demand substantial kernel-level and hardware-level optimizations to achieve competitive throughput. In particular, because Image-LoRA applies updates only to visual-token spans, designing an efficient inference pathway becomes crucial: in its current form, Image-LoRA must dynamically locate visual-token spans for every inference sample. This behavior differs fundamentally from Std-LoRA, which merges weights before inference without requiring such runtime checks.

Image-LoRA is most naturally applicable to transformer-based VLMs that represent images as explicit visual-token sequences. For architectures such as Llama-3.2-Vision *Meta* (2024), which integrate visual information via cross-attention directly into the text representation, Image-LoRA behaves much more like Std-LoRA, since the notion of a distinct visual-token span becomes less well-defined.

Regarding experimental scale, our current results are limited to on the order of a thousand samples due to constrained compute and the need to validate multiple components. A more comprehensive evaluation would ideally follow the large-scale setting of *Schulman and Lab* (2025), which employs extensive data and dense sweeps over experimental settings, such as LoRA-rank variations.

Moreover, a detailed analysis of how Image-LoRA reshapes attention patterns across heads and layers—and how these patterns differ from those induced by Std-LoRA—could clarify its underlying mechanisms and inform further improvements. Beyond that, an open question is whether the dynamics of these pattern changes during training differ between Image-LoRA and Std-LoRA.

Finally, understanding the interaction between Image-LoRA and head selection is an important avenue for future work. Our head-selection procedure is central to reducing the number of trainable parameters and, consequently, adapter-only FLOPs. It would be valuable to study how the selected heads evolve over the course of Image-LoRA training, particularly in web-scale settings where

different subsets of attention heads may be optimal at different training stages. Moreover, this paper does not systematically examine how different head-selection methods affect performance: we focus on a single, simple head-selection scheme, and alternative criteria may lead to substantially different accuracy–efficiency trade-offs.

## 6.6 Conclusion

We presented *Image-LoRA*, a simple PEFT recipe for transformer-based VLMs that restricts adaptation to visual tokens and a small subset of attention heads selected by a one-shot influence score, with a layer normalization to stabilize updates. Across screen-centric grounding and referring setups spanning a range of text:image token ratios, Image-LoRA matched or approached standard LoRA while using fewer trainable parameters and lower adapter-only FLOPs. It preserved text-only reasoning while showing signs of reduced reliance on language priors. Despite being evaluated on limited backbones and tasks, we hope this vision-focused PEFT recipe inspires broader exploration.

# CHAPTER 7

# Visual Test-time Scaling for GUI Agent Grounding

*The preceding chapters addressed structure in representations, supervision, diagnostics, and model adaptation. This final technical chapter considers* ***structure at test time****. In interactive GUI tasks, success hinges on a small set of relevant interface elements amid visual clutter. We introduce a test-time scaling strategy that dynamically focuses the model's attention on task-relevant regions, translating the structural assumption—"the answer lies in a localized area"—into improved action grounding.*

## 7.1 Introduction

Graphical user interface (GUI) agents have become increasingly pivotal in modern computing, powering applications ranging from automated web browsing to intuitive operating system navigation *Anderson et al.* (2018b); *Liu et al.* (2024c). With the proliferation of large-scale vision-language models (VLMs), researchers have sought to harness both textual and visual information to build more capable interactive systems *Anthropic* (2024). While many existing frameworks rely heavily on text-based reasoning *Zheng et al.* (2024); *Yang et al.* (2024) or simple visual grounding *Lu et al.* (2024); *Gou et al.* (2024), real-world GUIs often contain a substantial number of irrelevant

This chapter is based on work published at ICCV 2025 (*Luo et al.*, 2025b).

elements—such as menu bars, ads, and extraneous buttons—that can overwhelm purely textual or naive visual approaches. This mismatch between text-heavy inference and the visual complexity of GUIs leads to frequent errors (e.g., clicking the wrong button or navigating to an unintended section). Since these tasks are typically high-level, such low-level mistakes accumulate and ultimately result in higher failure rates and poorer overall performance.

Recent research on GUI agents typically falls into two main categories: those relying on textual cues for planning and reasoning, and those incorporating visual information through VLMs. Text-based approaches often generate text labels or bounding boxes for each visual element to guide agent actions *Zheng et al.* (2024); *Lu et al.* (2024). However, they can struggle when textual descriptions are ambiguous, incomplete, or fail to capture crucial visual features (e.g., floating windows), even when using accessibility trees. On the other hand, vision-based pipelines *Gou et al.* (2024); *Qin et al.* (2025) often rely heavily on the VLM's ability to ground visual elements. We observe that many errors arise from inadvertently clicking empty or incorrect interface components, underscoring the limitations of existing single-inference visual grounding methods using VLMs. Once an error occurs, there is no feedback loop to correct it, causing mistakes to compound throughout the process.

Motivated by these shortcomings, we propose a *visual test-time scaling* framework, RegionFocus, designed to narrow the GUI model's attention to salient interface regions when execution errors occur or other conditions are triggered (e.g., VLM self-judgment). Specifically, as illustrated in Figure 7.1(1), we leverage the VLM's capability to identify points of interest and combine this with bounding-box proposals generated either from fixed-ratio masks or segmentation models such as SAM *Kirillov et al.* (2023). For each sub-region, the agent independently predicts actions based solely on the local context (Figure 7.1(2)), subsequently aggregating the top candidate actions to form a refined, single-step response. Furthermore, interactions with web or OS interfaces allow our method to zoom into targeted areas, enhancing the resolution of selected regions for more careful examination. RegionFocus works as a modular plug-in for GUI agents without affecting the original workflow.

In order to keep track of regions visited by RegionFocus, we introduce an *image-as-map*

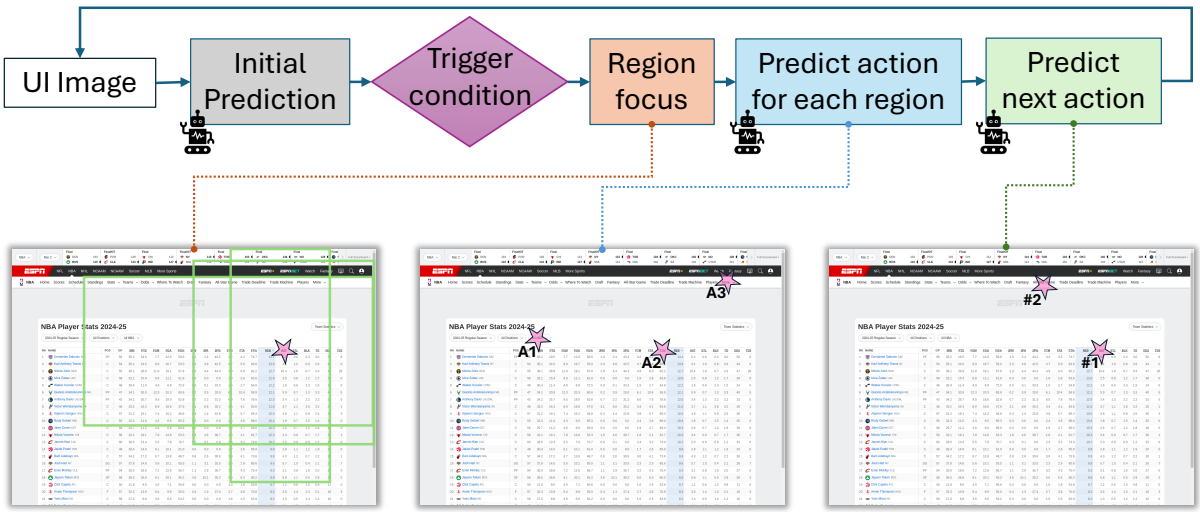



Figure 7.1: **Overview**. When GUI agents encounter execution errors, we instruct the model to focus on a specific point of interest and extract multiple sub-regions around this focal point (1). The agent then independently generates candidate actions for each sub-region. Actions that interact with a specific coordinate are marked with pink-star landmarks (e.g., "click") to visually indicate the relevant location, along with action numbers (2). We retain the pink-star landmarks annotated with history indices to track RegionFocus's past interactions for diverse exploration (3).

mechanism to record temporal information. In this approach, elements previously considered by the agent are annotated in the UI screenshot with visual landmarks (e.g., the pink-stars in Figure 7.1 (3)). These landmarks prevent the agent from revisiting regions it has already examined and guide the agent towards unexplored areas. These markers do not interfere with the model's regular inference process which leverages unaltered webpage screenshots, and are only used in the RegionFocus component. Once the agent navigates to a new page, all landmarks are refreshed to have a new RegionFocus history.

In addition to representing RegionFocus history, we also leverage image-as-map in the action aggregation process (Figure 7.1 (2)), where candidate actions are represented using landmarks for selecting an optimal action. We find image-as-map to be highly effective in representing both temporal information (e.g., previously visited regions) and spatial information (e.g., multiple action candidates) compared to alternative representations such as element coordinates represented in the

form of text alone. This is particularly crucial for distinguishing between screen elements in close proximity, which is challenging to reason about based on a text representation of the coordinates alone.

With our proposed visual test-time scaling framework, we help existing models— UI-TARS *Qin et al.* (2025) and QWen2.5-VL *Bai et al.* (2025) —achieve better performance in both web-based and desktop interfaces. In particular, we demonstrate substantial performance gains on benchmarks including *ScreenSpot-Pro Li et al.* (2025a) for OS-level GUI navigation, as well as *WebVoyager He et al.* (2024) for browser automation. Through our experiments, we show that even a simple fixed-ratio bounding-box generation approach yields pronounced improvements over baseline systems, underscoring the efficacy of focusing the model's attention on visually relevant regions. Our empirical studies further indicate that image-as-map consistently outperforms text-based representations for VLM agents. Overall, our findings highlight the value of visual test-time scaling as a simple yet powerful extension to existing VLM-based GUI agents.

## 7.2 Related Work

### 7.2.1 GUI Agents

Recent advancements in Large Language Models (LLMs) and Vision Language Models (VLMs) have significantly enhanced GUI automation, enabling agents to interact with diverse graphical environments through textual and visual modalities *Yao et al.* (2022); *Deng et al.* (2023); *Yan et al.* (2023); *Kim et al.* (2023); *Wang et al.* (2024c); *Gou et al.* (2024); *Liu et al.* (2024c); *Wang et al.* (2024b); *Pahuja et al.* (2025); *Trabucco et al.* (2025); *Murty et al.* (2025). Prior studies generally adopt two distinct approaches: (1) text-based reasoning, which leverages structured representations such as HTML or accessibility trees *Koh et al.* (2024); *Cao et al.* (2025); *Zhou et al.* (2024a), extracting structured interface information *Lu et al.* (2024); *Yu et al.* (2025) and supplementary textual details for input into LLMs/VLMs *Zheng et al.* (2024); and (2) vision-based inference, relying on VLMs to directly interpret GUI elements *Gou et al.* (2024); *Qin et al.* (2025). While

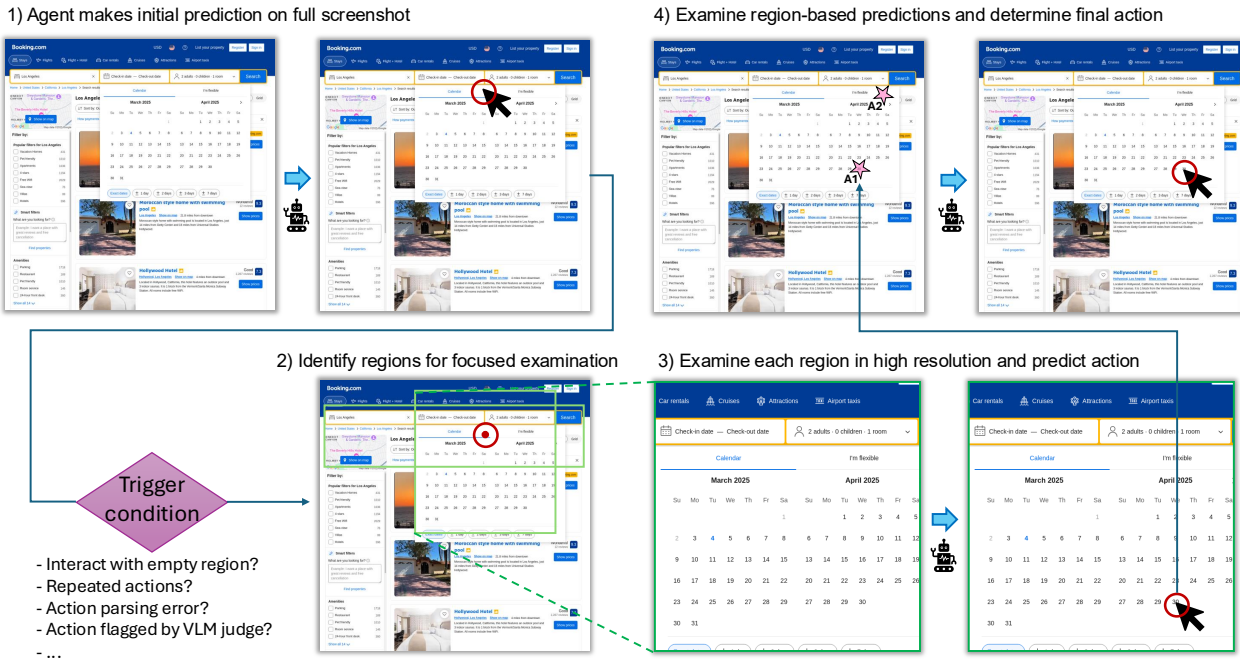



Figure 7.2: **Overview of integrating RegionFocus into GUI agent pipelines.** The standard inference process (blue arrow) takes input information and continuously predicts subsequent actions. When the GUI agent encounters errors, RegionFocus (green arrow) activates, proposing focal points to extract targeted sub-regions. Actions are then predicted individually for these sub-regions and aggregated into a single refined action for standard inference.

text-based techniques efficiently handle structured information, they often struggle with visually complex or ambiguous interfaces *Xie et al.* (2025); *He et al.* (2024); *Li et al.* (2025a), resulting in inaccuracies and reduced reliability. Similar observations have been reported in *Yang et al.* (2024). Conversely, visual grounding approaches may inadvertently interact with irrelevant or empty regions due to overly broad visual attention. In contrast to prior methods using entire interface screens as input *Shaw et al.* (2023); *Cheng et al.* (2024); *Hong et al.* (2024), our approach explicitly separates planning from visual grounding via a novel visual test-time scaling framework. This framework selectively targets salient GUI regions through precise bounding-box proposals and integrates an innovative “image-as-map" strategy, maintaining contextual coherence throughout interactions.

### 7.2.2 Test-Time Scaling in AI Agents

Test-time scaling involves dynamically adjusting computational resources during inference to enhance model performance *Wang et al.* (2022a); *Yao et al.* (2023b,a); *Tran Bao et al.* (2025). This approach allows AI agents to allocate additional processing power to challenging tasks, thereby improving decision-making and accuracy. Inspired by advancements in test-time scaling for LLMs, several studies have extended similar principles to GUI agents. For example, prior work leverage intermediate action histories *Zhang and Zhang* (2023), collect external information during inference *Nakano et al.* (2021) and incorporate reflection mechanisms into AI agents *Yu et al.* (2024b) during inference. Despite their success in improving performance, these methods do not utilize the unique advantages of visual information. We propose an approach toward visual test-time scaling, dynamically adjusting the image focus region and employ an “image-as-map" technique to encode history information for effective GUI agent inference.

### 7.2.3 Visual Image Attention

Visual image attention mechanisms have a rich history of enabling AI models to selectively focus on pertinent regions within visual inputs *Xu et al.* (2015); *Linsley et al.* (2018); *Yang et al.* (2018); *Farinhas et al.* (2021); *Guo et al.* (2022). Such mechanisms are especially critical for GUI-based agents, where accurately identifying and interacting with interface elements amidst visually cluttered environments is essential. Our proposed approach introduces a region-focused mechanism utilizing predefined bounding boxes that progressively refine their attention through history recording, incrementally concentrating focus more precisely on target elements. This iterative refinement shares conceptual similarities with prior recurrent models *Mnih et al.* (2014); *Cai and Vasconcelos* (2018); however, our method leverages VLMs to achieve refinement *Nasiriany et al.* (2024), uniquely integrating this process directly into GUI agents' test-time scaling. Accurately generating these attention regions directly via VLMs remains an open direction for future research.

## 7.3 Method

To address the limitations of current GUI agent frameworks, we propose a visual test-time scaling approach that enhances the robustness and accuracy of VLM agents interacting with complex graphical user interfaces. Unlike traditional methods, which uniformly treat all interface elements, our method dynamically adjusts the model's focus by selectively emphasizing visually salient regions whenever potential errors are detected. This approach significantly reduces misclicks and navigation mistakes.

Crucially, our framework operates entirely at inference time, enabling straightforward integration into existing VLM agents without requiring retraining or architectural modifications. In the following sections, we first describe the integration process for our pipeline with existing GUI agents, followed by a detailed explanation of the design principles and implementation of each component.

### 7.3.1 Overview

Figure 7.2 illustrates our proposed pipeline. On top of the standard interaction pipeline, our approach introduces an error-triggered refinement mechanism. Specifically, when the agent encounters a *trigger condition*—such as clicking on non-interactive elements (e.g., selecting empty space instead of the intended date option) or repeating unsuccessful actions (e.g., repeatedly typing "Los Angeles" without successfully clicking the correct button)—the *RegionFocus* component is activated.

During this refinement stage, the agent initially predicts a focal point near the intended target element. Based on this approximate location, it generates a bounding box likely to encapsulate the target element (Section 7.3.2). For each region defined by the bounding boxes, the agent independently predicts candidate actions. Finally, the agent aggregates a single action to be executed based on the predictions for each region.

Additionally, we maintain an annotated history of previously examined focal points on the same UI image using an *image-as-map* representation (Section 7.3.3). This visual history guides the agent

in avoiding redundant searches and helps it progressively focus on the correct target. Interactive predictions involving specific coordinates are visually marked (e.g., using pink-star landmarks) to help the model verify the correctness of its selections.

### 7.3.2 Visual Region Focus

The core of our pipeline is a dynamic visual zoom-in mechanism activated during inference. When initial action predictions result in errors—such as clicks on non-interactive or empty regions—the model dynamically adjusts its visual attention. Specifically, it refines attention by generating bounding-box proposals around visually salient regions, leading to more precise single-step actions.

**Trigger Condition** We define two primary types of triggering conditions. The first relies on environmental feedback obtained from direct interaction with dynamic GUI environments, such as interactive webpages. Errors, like clicks on non-interactive elements, can be easily detected based on environment feedback (e.g., webpage change). The second type of trigger occurs in cases where environmental interaction isn't possible and we are dealing with static screenshots (e.g., ScreenSpot-Pro *Li et al.* (2025b) scenario). Here, the VLM evaluates predicted actions, shifting its role from merely making predictions to explicitly evaluating the correctness of actions. Although the VLM's judgments aren't always perfectly accurate, they significantly help identify and mitigate errors shown in our experiments.

**Bounding-box Proposal** Empirically, VLM agents reliably produce focal points near target elements but struggle to directly predict accurate bounding boxes *Feng et al.* (2025). Instead, we derive bounding boxes from these focal points rather than predicting them directly. Although advanced segmentation models (e.g., SAM (*Kirillov et al.*, 2023)) could provide more precise bounding boxes, we currently focus exclusively on leveraging the GUI agent itself. This ensures that future enhancements in GUI agents directly benefit our approach.

We employ a heuristic approach, defining bounding boxes with fixed dimensions and aspect ratios (e.g., 0.5 width by 0.5 height) centered around the focal point. If a bounding box exceeds

the image boundary, it is adjusted to remain fully within the screenshot. This simple yet effective strategy strikes a good balance between accuracy and computational efficiency.

**Action Candidate Prediction** Given the regions identified by the bounding boxes extracted from the previous stage, the agent predicts an action for each region. If the agent can interact with the environment (e.g., in the case of a webpage), a zoomed-in, high resolution view of the region is provided to the agent. In this case, at least one side of the region can be made to match the original full image resolution via zooming in. If environment interaction is not possible, we simply crop the region from the initial image and upsample it for prediction.

**Action Aggregation** After the GUI agent independently analyzes each bounding-box proposal and generates candidate actions, we prompt the model to select a single action from all candidates based on the overall task objective, serving as the next step in the inference pipeline. For coordinate-based actions (e.g., “Click (x, y)" or “Scroll (x, y) down"), we visually mark the candidate action coordinates on the snapshot along with their action numbers, as illustrated in Figure 7.1 (2).[1] This process significantly reduces the model’s workload by simplifying the mapping and interpretation of textual coordinates onto the image. Empirically, we observe that incorporating these visual markers leads the model to select action candidates more accurately.

### 7.3.3 Image-as-Map

When RegionFocus is triggered multiple times for a given UI image, it is important for the agent to generate diverse focal points and avoid revisiting previously explored regions. Initially, we attempted to represent the region focus history using textual coordinates but found this approach ineffective, as the agent often revisited similar focal points despite explicit prompts to avoid them.

To address this, we propose an image-as-map representation, where we visually encode previously examined focal points as pink-star landmarks annotated with historical indices directly onto UI snapshots, as shown in Figure 7.3 & Figure 7.1 (3), to reflect temporal information. Unlike

[1]Landmark annotations are only used for actions involving interaction with specific points and elements in the current view.

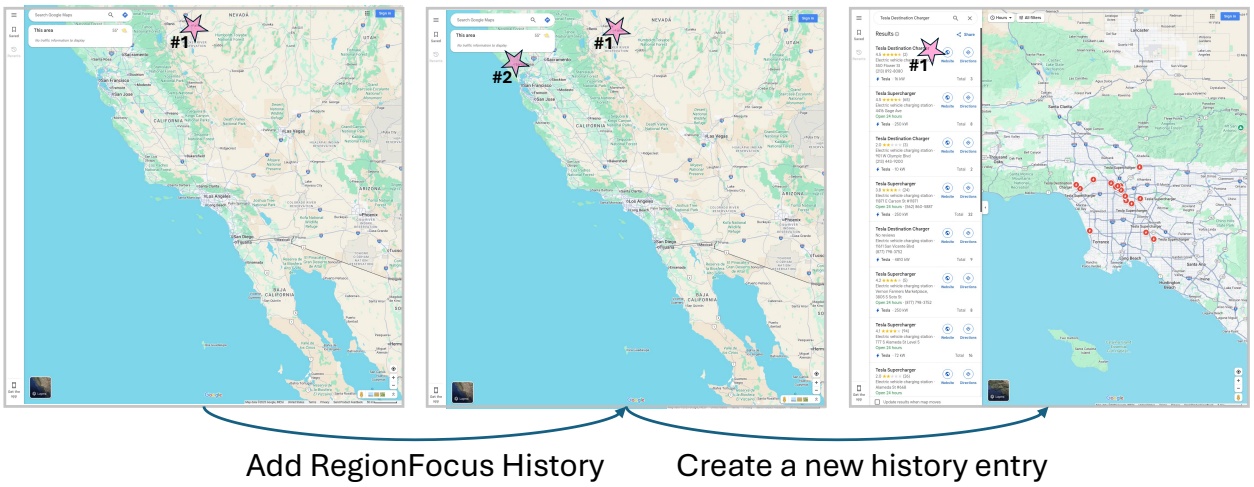



Figure 7.3: **Image-as-Map records temporal information.** We place numbered pink stars in the image as visual landmarks to indicate previously focused points. Each time a RegionFocus attempt fails (i.e., action takes no effect), we add a new pink star at the attempted location. Once an action successfully takes effect, we refresh the history and remove any existing landmarks.

| Agent Model | Development | | | Creative | | | CAD | | | Scientific | | | Office | | | OS | | | Avg | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | text | icon | avg | text | icon | avg | text | icon | avg | text | icon | avg | text | icon | avg | text | icon | avg | text | icon | avg |
| QwenVL-7B | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.7 | 0.0 | 0.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.0 | 0.1 |
| GPT-4o | 1.3 | 0.0 | 0.7 | 1.0 | 0.0 | 0.6 | 2.0 | 0.0 | 1.5 | 2.1 | 0.0 | 1.2 | 1.1 | 0.0 | 0.6 | 0.0 | 0.0 | 0.0 | 1.3 | 0.0 | 0.8 |
| SeeClick | 0.6 | 0.0 | 0.3 | 1.0 | 0.0 | 0.6 | 2.5 | 0.0 | 1.9 | 3.5 | 0.0 | 2.0 | 1.1 | 0.0 | 0.5 | 2.8 | 0.0 | 1.5 | 1.8 | 0.0 | 1.1 |
| Qwen2-VL-7B | 2.6 | 0.0 | 1.3 | 1.5 | 0.0 | 0.9 | 0.5 | 0.0 | 0.4 | 6.3 | 0.0 | 3.5 | 3.4 | 1.9 | 3.0 | 0.9 | 0.0 | 0.5 | 2.5 | 0.2 | 1.6 |
| OS-Atlas-4B | 7.1 | 0.0 | 3.7 | 3.0 | 1.4 | 2.3 | 2.0 | 0.0 | 1.5 | 9.0 | 5.5 | 7.5 | 5.1 | 3.8 | 4.4 | 5.6 | 0.0 | 3.1 | 5.0 | 1.7 | 3.7 |
| ShowUI-2B | 16.9 | 1.4 | 9.4 | 9.1 | 0.0 | 5.3 | 2.5 | 0.0 | 1.9 | 13.2 | 7.3 | 10.6 | 15.3 | 7.5 | 13.5 | 10.3 | 2.2 | 6.6 | 10.8 | 2.6 | 7.7 |
| CogAgent-18B | 14.9 | 0.7 | 8.0 | 9.6 | 0.0 | 5.6 | 7.1 | 3.1 | 6.1 | 22.2 | 1.8 | 13.4 | 13.0 | 0.0 | 6.5 | 5.6 | 0.0 | 3.1 | 12.0 | 0.8 | 7.7 |
| Aria-UI | 16.2 | 0.0 | 8.4 | 23.7 | 2.1 | 14.7 | 7.6 | 1.6 | 6.1 | 27.1 | 6.4 | 18.1 | 20.3 | 1.9 | 16.1 | 4.7 | 0.0 | 2.6 | 17.1 | 2.0 | 11.3 |
| UGround-7B | 26.6 | 2.1 | 14.7 | 27.3 | 2.8 | 17.0 | 14.2 | 1.6 | 11.1 | 31.9 | 2.7 | 19.3 | 31.6 | 11.3 | 27.9 | 17.8 | 0.0 | 9.7 | 25.0 | 2.8 | 16.5 |
| Claude Comp.Use | 22.0 | 3.9 | 12.6 | 25.9 | 3.4 | 16.8 | 14.5 | 3.7 | 11.9 | 33.9 | 15.8 | 25.8 | 30.1 | 16.3 | 26.2 | 11.0 | 4.5 | 8.1 | 23.4 | 7.1 | 17.1 |
| OS-Atlas-7B | 33.1 | 1.4 | 17.7 | 28.8 | 2.8 | 17.9 | 12.2 | 4.7 | 10.3 | 37.5 | 7.3 | 24.4 | 33.9 | 5.7 | 27.4 | 27.1 | 4.5 | 16.8 | 28.1 | 4.0 | 18.9 |
| UGround-V1-7B | - | - | 35.5 | - | - | 27.8 | - | - | 13.5 | - | - | 38.8 | - | - | 48.8 | - | - | 26.1 | - | - | 31.1 |
| UI-TARS-7B | 58.4 | 12.4 | 36.1 | 50.0 | 9.1 | 32.8 | 20.8 | **9.4** | 18.0 | 63.9 | **31.8** | 50.0 | 63.3 | 20.8 | 53.5 | 30.8 | 16.9 | 24.5 | 47.8 | 16.2 | 35.7 |
| + *RegionFocus* | **59.7** | **15.9** | **38.5** | **59.6** | **11.9** | **39.6** | **30.5** | 7.8 | **24.9** | **67.4** | 30.0 | **51.2** | **69.5** | **30.2** | **60.4** | **45.8** | **21.3** | **34.7** | **55.2** | **18.7** | **41.2** |
| UI-TARS-72B | 63.0 | 17.3 | 40.8 | 57.1 | 15.4 | 39.6 | 18.8 | 12.5 | 17.2 | 64.6 | 20.9 | 45.7 | 63.3 | 26.4 | 54.8 | 42.1 | 15.7 | 30.1 | 50.9 | 17.5 | 38.1 |
| + *RegionFocus* | **72.1** | **26.9** | **50.2** | **68.7** | **22.4** | **49.3** | **35.5** | **25.0** | **33.0** | **77.1** | **30.9** | **57.1** | **72.9** | **45.3** | **66.5** | **63.6** | **27.0** | **46.9** | **64.0** | **28.0** | **50.2** |
| Qwen2.5-VL-7B | 45.5 | 1.4 | 24.1 | 32.8 | **6.3** | 21.7 | 22.3 | **6.2** | 18.4 | 50.7 | 7.3 | 31.9 | 52.5 | 15.1 | 43.9 | 36.4 | 10.1 | 24.5 | 39.3 | 6.6 | 26.8 |
| + *RegionFocus* | **53.2** | **3.4** | **29.1** | **42.9** | 4.9 | **27.0** | **28.4** | 3.1 | **22.2** | **56.9** | **10.9** | **37.0** | **59.9** | **24.5** | **51.7** | **41.1** | **15.7** | **29.6** | **46.6** | **8.8** | **32.1** |
| Qwen2.5-VL-72B | 66.2 | 13.8 | 40.8 | 64.6 | 15.4 | 44.0 | 47.7 | 12.5 | 39.1 | 78.5 | 29.1 | 57.1 | 74.6 | 37.7 | 66.1 | 60.7 | 22.5 | 43.4 | 64.9 | 20.2 | 47.8 |
| + *RegionFocus* | **75.3** | **25.5** | **51.2** | **76.3** | **30.8** | **57.2** | **71.6** | **28.1** | **60.9** | **87.5** | **39.1** | **66.5** | **87.0** | **60.4** | **80.9** | **74.8** | **36.0** | **57.1** | **78.6** | **34.1** | **61.6** |

Table 7.1: **Comparison of various models on ScreenSpot-Pro.** *Li et al.* (2025b). Our proposed RegionFocus helps the UI-TARS-72B *Qin et al.* (2025) model achieve a 31.8% improvement, while Qwen2.5-VL-72B *Bai et al.* (2025) sees a 28.9% gain, thereby achieving state-of-the-art performance. Additionally, integrating RegionFocus into UI-TARS-7B allows it to surpass the performance of the substantially larger UI-TARS-72B model.

text-based histories, this visual representation more effectively conveys temporal information, allowing the agent to reason directly over the image and avoid revisiting the same areas. Note that the

visual landmark annotation process used for action aggregation (Figure 7.1 (2)) is also a form of the image-as-map strategy, but capturing spatial information. Landmark-annotated snapshots are used only in the RegionFocus process, whereas the original inference pipeline (e.g., for the initial action prediction) receives unaltered UI images. We maintain these highlighted landmarks on the page until an action takes effect (i.e., causes a meaningful state change), at which point the history is refreshed. Our empirical results show that this image-as-map strategy consistently outperforms text-based methods, especially in more complex, multi-step GUI tasks. We also observe that it helps the agent distinguish between two GUI elements that are very close to each other, as shown in Figure 7.7 (2).

## 7.4 Experiments

In the experimental sections below, we integrate our proposed pipeline with UI-TARS *Qin et al.* (2025) and Qwen2.5-VL *Bai et al.* (2025), two recently proposed GUI agent models that autonomously interacts with GUI screenshots, which have demonstrated exceptional performance in various GUI tasks. We evaluate our pipeline across both OS operation tasks and Web Interaction. For our main experiments, we adopt a fixed bounding box approach (Section 7.3.2), using only the agent model itself. We also include ablation studies using SAM *Kirillov et al.* (2023), which provides more accurate bounding boxes. Additional experimental details are provided in Appendix F.2.

### 7.4.1 ScreenSpot-Pro

ScreenSpot-Pro *Li et al.* (2025b) is a benchmark designed to evaluate GUI grounding in complex, high-resolution desktop environments (typically larger than $3k \times 2k$). It includes expert-annotated tasks across 23 applications spanning five functional domains (Development, Creative, CAD, Scientific, Office) and three operating systems. Tasks are categorized into text-based and icon-based grounding challenges. Grounding accuracy is measured by verifying if the model-predicted location falls within the bounding box of the target element.

Because ScreenSpot-Pro only provides static screenshots—without an OS environment for

| Agent Model | Allrecipes | Amazon | Apple | ArXiv | GitHub | Booking | ESPN | Coursera |
|---|---|---|---|---|---|---|---|---|
| Claude | 45.9%$_{\pm 3.4\%}$ | 58.6%$_{\pm 4.2\%}$ | 58.1%$_{\pm 4.0\%}$ | 55.0%$_{\pm 7.0\%}$ | 56.9%$_{\pm 1.4\%}$ | 19.0%$_{\pm 1.3\%}$ | 46.2%$_{\pm 1.3\%}$ | 68.2%$_{\pm 1.3\%}$ |
| GPT-4o | 56.3%$_{\pm 1.3\%}$ | 53.7%$_{\pm 2.5\%}$ | 56.6%$_{\pm 1.3\%}$ | 60.5%$_{\pm 0.0\%}$ | 57.7%$_{\pm 3.7\%}$ | 43.9%$_{\pm 3.5\%}$ | 44.0%$_{\pm 2.7\%}$ | 65.1%$_{\pm 2.8\%}$ |
| WebVoyager | 51.1%$_{\pm 2.2\%}$ | 52.9%$_{\pm 1.4\%}$ | 62.8%$_{\pm 2.3\%}$ | 52.0%$_{\pm 1.3\%}$ | 59.3%$_{\pm 3.7\%}$ | 32.6%$_{\pm 2.7\%}$ | 47.0%$_{\pm 1.3\%}$ | 57.9%$_{\pm 2.7\%}$ |
| Qwen2.5-VL-7B | 47.2%$_{\pm 4.4\%}$ | 49.1%$_{\pm 7.5\%}$ | 47.3%$_{\pm 2.5\%}$ | 14.9%$_{\pm 2.1\%}$ | 23.9%$_{\pm 4.8\%}$ | 10.0%$_{\pm 0.9\%}$ | 39.2%$_{\pm 7.2\%}$ | 46.4%$_{\pm 0.3\%}$ |
| + *RegionFocus* | **49.2%**$_{\pm 7.2\%}$ | **53.6%**$_{\pm 4.4\%}$ | **67.1%**$_{\pm 3.6\%}$ | **51.7%**$_{\pm 1.3\%}$ | **35.0%**$_{\pm 0.3\%}$ | **30.0%**$_{\pm 0.\%}$ | **39.6%**$_{\pm 6.4\%}$ | **71.1%**$_{\pm 4.8\%}$ |
| Qwen2.5-VL-72B | 28.6%$_{\pm 1.5\%}$ | 56.2%$_{\pm 4.3\%}$ | 53.0%$_{\pm 0.9\%}$ | 32.6%$_{\pm 2.4\%}$ | 61.5%$_{\pm 4.3\%}$ | 24.8%$_{\pm 2.9\%}$ | 49.8%$_{\pm 3.8\%}$ | 72.9%$_{\pm 1.9\%}$ |
| + *RegionFocus* | **43.4%**$_{\pm 1.3\%}$ | **60.6%**$_{\pm 1.5\%}$ | **69.7%**$_{\pm 4.0\%}$ | **45.4%**$_{\pm 2.0\%}$ | **67.3%**$_{\pm 3.8\%}$ | **37.3%**$_{\pm 2.9\%}$ | **59.8%**$_{\pm 1.4\%}$ | **78.2%**$_{\pm 1.4\%}$ |
| Qwen2.5-VL-72B RF-V5 | 68.4%$_{\pm nan\%}$ | 50.0%$_{\pm nan\%}$ | 66.7%$_{\pm nan\%}$ | 66.7%$_{\pm nan\%}$ | 61.5%$_{\pm nan\%}$ | 45.0%$_{\pm nan\%}$ | 30.0%$_{\pm nan\%}$ | 87.5%$_{\pm nan\%}$ |
| Qwen2.5-VL-72B RF-V5 multi-image | 56.4%$_{\pm nan\%}$ | 55.3%$_{\pm nan\%}$ | 81.8%$_{\pm nan\%}$ | 57.1%$_{\pm nan\%}$ | 62.5%$_{\pm nan\%}$ | 30.0%$_{\pm nan\%}$ | 40.0%$_{\pm nan\%}$ | 87.5%$_{\pm nan\%}$ |
| UI-TARS-7B | 17.8%$_{\pm 1.3\%}$ | 30.9%$_{\pm 1.4\%}$ | 17.1%$_{\pm 1.3\%}$ | 20.9%$_{\pm 2.3\%}$ | 32.5%$_{\pm 2.5\%}$ | 7.6%$_{\pm 1.3\%}$ | 45.0%$_{\pm 1.3\%}$ | 70.7%$_{\pm 1.2\%}$ |
| + *RegionFocus* | **35.6%**$_{\pm 2.2\%}$ | **39.0%**$_{\pm 2.4\%}$ | **31.8%**$_{\pm 2.7\%}$ | **37.2%**$_{\pm 0.0\%}$ | **58.1%**$_{\pm 1.5\%}$ | **15.2%**$_{\pm 1.3\%}$ | **62.8%**$_{\pm 0.0\%}$ | **74.2%**$_{\pm 1.4\%}$ |
| UI-TARS-72B | 16.2%$_{\pm 2.6\%}$ | 38.6%$_{\pm 2.0\%}$ | 45.9%$_{\pm 1.2\%}$ | **64.9%**$_{\pm 4.1\%}$ | 43.2%$_{\pm 2.0\%}$ | 40.2%$_{\pm 2.3\%}$ | 47.2%$_{\pm 3.8\%}$ | 58.9%$_{\pm 1.6\%}$ |
| + *RegionFocus* | **57.5%**$_{\pm 1.1\%}$ | **55.8%**$_{\pm 1.4\%}$ | **52.6%**$_{\pm 1.3\%}$ | 50.3%$_{\pm 3.6\%}$ | **58.9%**$_{\pm 3.7\%}$ | **44.7%**$_{\pm 2.6\%}$ | **69.7%**$_{\pm 1.3\%}$ | **82.2%**$_{\pm 1.5\%}$ |

| Agent Model | Cambridge Dictionary | BBC News | Google Flights | Google Map | Google Search | Huggingface | Wolfram Alpha | Overall |
|---|---|---|---|---|---|---|---|---|
| Claude | 71.3%$_{\pm 3.6\%}$ | 66.7%$_{\pm 4.8\%}$ | 15.1%$_{\pm 5.5\%}$ | 55.3%$_{\pm 1.4\%}$ | 72.9%$_{\pm 1.3\%}$ | 53.5%$_{\pm 4.7\%}$ | 51.5%$_{\pm 5.4\%}$ | 52.8%$_{\pm 1.4\%}$ |
| GPT-4o | 82.2%$_{\pm 1.3\%}$ | 54.8%$_{\pm 2.4\%}$ | 28.6%$_{\pm 0.0\%}$ | 56.9%$_{\pm 2.8\%}$ | 63.6%$_{\pm 1.3\%}$ | 42.6%$_{\pm 3.6\%}$ | 65.2%$_{\pm 2.2\%}$ | 55.5%$_{\pm 0.8\%}$ |
| WebVoyager | 71.3%$_{\pm 1.3\%}$ | 60.3%$_{\pm 2.8\%}$ | 51.6%$_{\pm 1.4\%}$ | 64.3%$_{\pm 2.8\%}$ | 77.5%$_{\pm 2.7\%}$ | 55.8%$_{\pm 2.3\%}$ | 60.9%$_{\pm 2.2\%}$ | 57.1%$_{\pm 0.2\%}$ |
| Qwen2.5-VL-7B | **21.1%**$_{\pm 4.8\%}$ | 45.1%$_{\pm 3.0\%}$ | 10.0%$_{\pm 0.9\%}$ | **30.2%**$_{\pm 1.5\%}$ | 10.0%$_{\pm 1.3\%}$ | 41.4%$_{\pm 0.3\%}$ | **51.3%**$_{\pm 3.8\%}$ | 32.5%$_{\pm 1.3\%}$ |
| + *RegionFocus* | 17.3%$_{\pm 0.3\%}$ | **52.9%**$_{\pm 0.9\%}$ | **12.8%**$_{\pm 4.8\%}$ | 17.1%$_{\pm 1.2\%}$ | **18.3%**$_{\pm 2.4\%}$ | **60.0%**$_{\pm 2.8\%}$ | 40.8%$_{\pm 1.2\%}$ | **41.1%**$_{\pm 1.2\%}$ |
| Qwen2.5-VL-72B | 63.7%$_{\pm 1.3\%}$ | 45.9%$_{\pm 1.6\%}$ | 17.7%$_{\pm 1.5\%}$ | 31.2%$_{\pm 1.5\%}$ | 11.5%$_{\pm 0.0\%}$ | 38.3%$_{\pm 2.9\%}$ | 48.9%$_{\pm 2.5\%}$ | 42.4%$_{\pm 0.5\%}$ |
| + *RegionFocus* | **68.9%**$_{\pm 1.3\%}$ | **54.4%**$_{\pm 2.9\%}$ | **34.6%**$_{\pm 4.4\%}$ | **42.2%**$_{\pm 1.5\%}$ | **20.3%**$_{\pm 0.0\%}$ | **51.5%**$_{\pm 4.9\%}$ | **56.0%**$_{\pm 1.1\%}$ | **52.7%**$_{\pm 1.1\%}$ |
| Qwen2.5-VL-72B RF-v5 | 7.0%$_{\pm nan\%}$ | 57.1%$_{\pm nan\%}$ | 51.3%$_{\pm nan\%}$ | 47.4%$_{\pm nan\%}$ | 20.0%$_{\pm nan\%}$ | 51.4%$_{\pm nan\%}$ | 63.0%$_{\pm nan\%}$ | 51.5%$_{\pm nan\%}$ |
| Qwen2.5-VL-72B RF-v5 Multi | 9.3%$_{\pm nan\%}$ | 60.0%$_{\pm nan\%}$ | 46.2%$_{\pm nan\%}$ | 57.9%$_{\pm nan\%}$ | 7.5%$_{\pm nan\%}$ | 54.3%$_{\pm nan\%}$ | 56.5%$_{\pm nan\%}$ | 50.8%$_{\pm nan\%}$ |
| UI-TARS-7B | **57.1%**$_{\pm 1.3\%}$ | 41.3%$_{\pm 2.7\%}$ | 10.3%$_{\pm 1.4\%}$ | 17.5%$_{\pm 0.0\%}$ | 49.6%$_{\pm 1.3\%}$ | 40.7%$_{\pm 1.4\%}$ | 38.4%$_{\pm 1.3\%}$ | 33.2%$_{\pm 0.5\%}$ |
| + *RegionFocus* | 55.8%$_{\pm 0.0\%}$ | **52.4%**$_{\pm 2.4\%}$ | **28.6%**$_{\pm 4.8\%}$ | **29.3%**$_{\pm 2.4\%}$ | **60.5%**$_{\pm 2.3\%}$ | **45.7%**$_{\pm 1.3\%}$ | **44.2%**$_{\pm 1.3\%}$ | **44.7%**$_{\pm 0.5\%}$ |
| UI-TARS-72B | 72.9%$_{\pm 1.4\%}$ | 39.3%$_{\pm 3.4\%}$ | 33.9%$_{\pm 2.1\%}$ | 27.2%$_{\pm 5.2\%}$ | 60.6%$_{\pm 1.2\%}$ | 24.9%$_{\pm 1.6\%}$ | 48.3%$_{\pm 2.3\%}$ | 44.1%$_{\pm 0.5\%}$ |
| + *RegionFocus* | **75.9%**$_{\pm 1.3\%}$ | **49.8%**$_{\pm 3.6\%}$ | **49.9%**$_{\pm 2.1\%}$ | **59.9%**$_{\pm 1.4\%}$ | **65.8%**$_{\pm 1.2\%}$ | **59.6%**$_{\pm 1.3\%}$ | **60.2%**$_{\pm 1.3\%}$ | **59.5%**$_{\pm 0.1\%}$ |

Table 7.2: **Comparison of various models on WebVoyager** *He et al.* (2024). For each automatic evaluation, we run GPT evaluator three times to calculate the performance mean and standard deviation. We evaluated our method on UI-TARS *Qin et al.* (2025) and Qwen2.5-VL *Bai et al.* (2025), consistently observing performance improvements. The performance on Google Search was significantly affected by bot blocking - See text for details.

interaction—we employ a VLM-based judge to trigger RegionFocus. Specifically, when the agent predicts a point, we will highlight that point in the input screenshot with pink-star landmarks and ask the model itself to assess the correctness of that point. If deemed incorrect, we initiate the RegionFocus process.

**Results** In Table 7.1, we summarize the reported grounding accuracy of various methods evaluated on ScreenSpot-Pro. For fair comparisons, we employ the official test code released by ScreenSpot-Pro for evaluation. We report the original numbers from the UI-TARS paper in Table 7.1.

Table 7.1 shows that RegionFocus consistently improves performance across all categories for both text and icon grounding compared to the base model. UI-TARS-72B + RegionFocus achieves a 31.7% improvement over the base UI-TARS-72B model. Moreover, the UI-TARS-7B + RegionFocus variant outperforms the UI-TARS-72B model overall, demonstrating the effectiveness

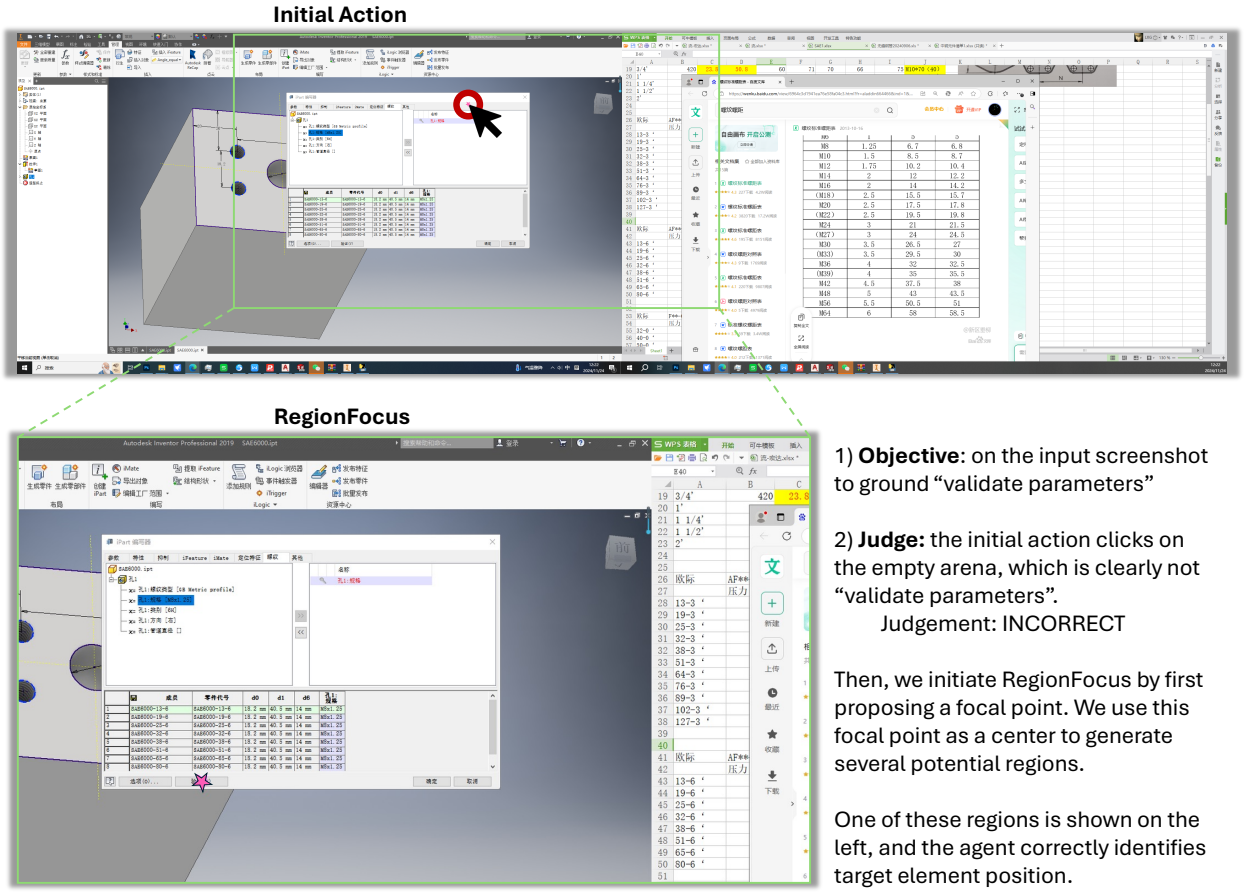



Figure 7.4: **Qualitative results - Screenspot-Pro.** In this example, the agent successfully rejects the initial action via self-VLM-judge and proposes a correct grounding point based on the zoomed-in view. More qualitative results are listed in Appendix B.

of our approach. Furthermore, RegionFocus further helps Qwen2.5-VL-72B achieve the state-of-the-art performance of 61.6%.

Figure 7.4 illustrates our pipeline for ScreenSpot-Pro. We first ask the agent to judge the initial action prediction. If it is incorrect, we initiate RegionFocus by zooming into the regions predicted by the agent, predicting actions within each region, and finally aggregating all actions into a single outcome. We used the same VLM to both predict the action and evaluate its correctness, and found that although it may initially generate incorrect coordinates, it can still reliably assess whether the click points are correct with the help of image-as-map—enabling enabling those predictions to be corrected by RegionFocus through prediction over the zoomed-in images. This generation-verification gap has also been noted in recent literature *Cook* (2023); *Swamy et al.*

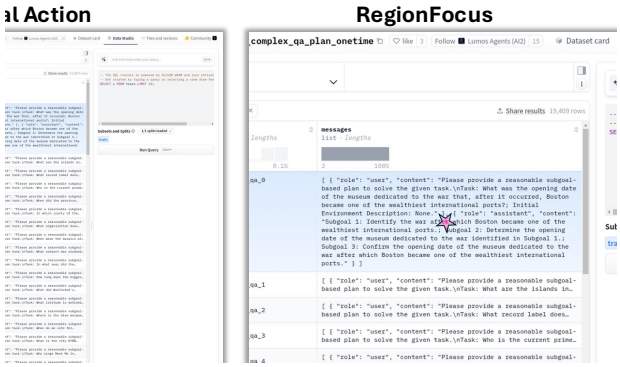



Figure 7.5: **RegionFocus.** We illustrate how RegionFocus reduces background noise by emphasizing salient regions of an image. The mouse pointer indicates the agent's initial action prediction, which is suboptimal. The second image is zoomed in, reducing distracting details and allowing the agent to focus on the relevant information.

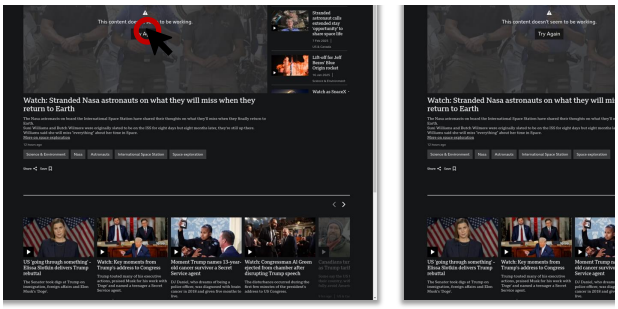



Figure 7.6: **Image-as-Map.** We show that action aggregation, enhanced by the proposed image-as-map, effectively distinguishes subtle coordinate differences between elements. Star landmarks are generated during RegionFocus prior to aggregation; the two landmarks at the top left correspond to the home and search buttons.

(2025).

### 7.4.2 WebVoyager

WebVoyager *He et al.* (2024) is a benchmark designed to evaluate autonomous web agents' capabilities in performing open-ended tasks through multimodal interactions with real-world websites. Distinct from previous web agent benchmarks, WebVoyager comprises 643 tasks across 15 popular, real-world websites such as Amazon, Apple, ArXiv, and Google Maps. In this scenario, the agent actively interacts with the web environment. We employ a Playwright-controlled Chrome browser to navigate webpages, with the VLM agent determining the appropriate action based on each webpage screenshot. After task execution, we use the official evaluation setting, where a GPT-based judge reviews the last 15 screenshots along with an optional textual response to determine whether the task has been successfully accomplished.

**Results** Table 7.2 presents the task success rates of various web agents evaluated on the WebVoyager benchmark. RegionFocus consistently improves performance across all types of websites—including "Booking" and "Search"—highlighting the effectiveness of integrating Region-

Focus into the GUI agent for web browsing. It also brings consistent improvements over two open-source model, UI-TARS and Qwen2.5-VL. We note that our model performance was impacted by online interaction constraints—such as bot blocking and intermittent VPN issues. For example, "Google-Search" performance dropped noticeably after Google recently tightened its bot detection systems and access controls; our UI-TARS results were obtained prior to this, whereas Qwen2.5-VL results were obtained afterward. While addressing these constraints can further improve the model's overall performance, bot-blocking and non-stationarity remain inherent challenges when evaluating agents on real-world websites.

We present qualitative examples from WebVoyager in Figures 7.5 and 7.6. Figure 7.5 shows RegionFocus zooming in on the sub-region of interest, enlarging key content and simplifying target localization. Figure 7.6 demonstrates a correction scenario: after initially clicking an unrelated element, our pipeline proposes two closely positioned buttons, and the image-as-map mechanism helps distinguish between these nearly identical elements with slight coordinate differences. More analysis are included in Appendix F.1.

### 7.4.3 Ablation Studies & Analysis

We conduct ablation studies on the entire pipeline, including our "image-as-map" design choice and the use of a predefined bounding box based on the point predicted by the agent. We also demonstrate that by leveraging SAM *Kirillov et al.* (2023) and increasing the number of trajectory steps, performance can be further improved. Due to limited computation, we employed the UI-TARS-7B-DPO model for these ablations on a subset of the WebVoyager benchmark. The results are shown in Table 7.3, where "image-as-map" and "Fixed-BBox" refers to our same 7B model configuration with RegionFocus enabled, a maximum limit of 100 action steps, and predefined bounding box with fixed aspect ratios.

**Text-based RegionFocus History Representation** By using image-as-map, we can directly provide visual location information to the agent, helping it distinguish

| **Agent Model** | Overall |
|---|---|
| Image-as-Map | 43.2 |
| Text-as-History | 37.2 |
| Fixed-BBox | 43.2 |

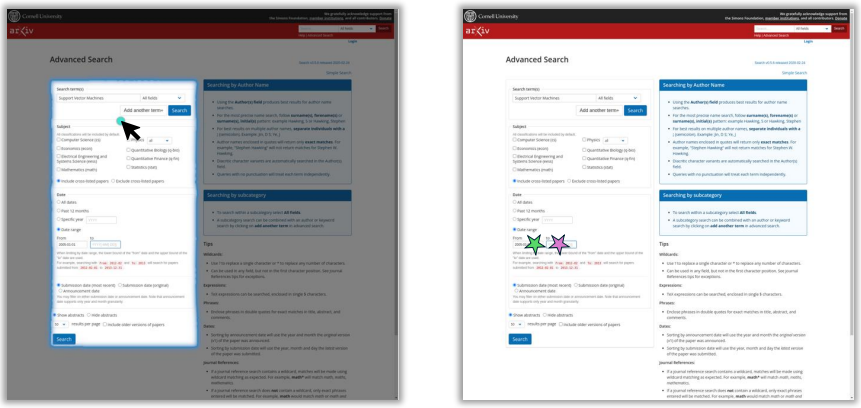



Figure 7.7: **Ablation studies.** (1) RegionFocus can natively use the SAM to generate bounding boxes. (2) how image-as-map helps highlight subtle differences of different GUI elements.

even minor differences and thereby enhancing its perception. For instance, as shown in Figure 7.7 (2), we color-code click points in the image to denote the image-as-map mechanism, while the corresponding coordinates are listed in the text box on the right. Notably, although the textual coordinate difference is within only five pixels, the resulting action can vary significantly. Using a purely text-based representation for both history tracking and action aggregation leads to notable degradation compared to our image-as-map approach (Text-as-History vs. Image-as-Map in Table 7.3).

**Proposing regions directly** In our pipeline (Section 7.3), instead of directly predicting bounding boxes, we prompt the agent model to first identify a point of interest and then generate a predefined bounding box around it. This design choice addresses the agent's difficulty in accurately predicting bounding boxes. To validate this, we conducted an experiment requiring the model to predict both upper-left and bottom-right coordinates for cropping the UI image. As indicated by the "Predict-Region" results, this approach significantly reduced performance.

**SAM** Our pipeline can naturally leverage segmentation models that take point inputs as indicator, such as SAM *Kirillov et al.* (2023). As shown in Figure 7.7 (1), even when the model provides a point in a non-interactive empty area, SAM successfully generates a bounding box covering the correct region—demonstrating its suitability. The effectiveness of incorporating point-based segmentation models into RegionFocus is further validated by the "SAM" results in Table 7.3.

**Test-time Thinking Budget** In our main experiments, we limited the maximum number of inference steps to 100. We investigate how performance changes when varying the inference-step limit above and below the default of 100 steps. Here, inference steps refer to actual browser-interactive actions, excluding RegionFocus overhead. We evaluated limits of 10, 50, and 300 steps, using the UI-TARS-7B model. As shown in Figure 7.8, raising the limit to 300 steps improved performance from 43.2 to 45.3, although most trajectories terminated before reaching this threshold. However, incremental benefits gradually diminish with higher inference-step limits.

**Runtime Analysis** For each trigger, the VLM performs one inference for focal-point proposal, four inferences for region action prediction (which can be executed in parallel), and one inference for action aggregation. The four region action prediction inferences result from adopting four predefined bounding boxes around each predicted focal point, a hyper-parameter. For the interactive environment (WebVoyager), *RegionFocus* occurs in 61.7% of all trajectories; among these, it is triggered an average of 5.84 times, incurring an average overhead of 66.8% per trajectory. Notably, 32.3% of triggered cases occur only once, with this single trigger yielding an 83.7% improvement in success. For the static environment (ScreenSpot-Pro), *RegionFocus* occurs in 60.2% and 33% of cases for the 72B and 7B models, respectively. The 72B model incurs overheads of 180% when region predictions are executed in parallel, and 360% if executed sequentially. Regarding memory usage, *RegionFocus* requires no additional GPU memory if region predictions are performed sequentially without employing SAM *Kirillov et al.* (2023). Whether region action predictions should be executed sequentially or in parallel, the number of regions to zoom into, and whether to utilize SAM depend on the specific use case.

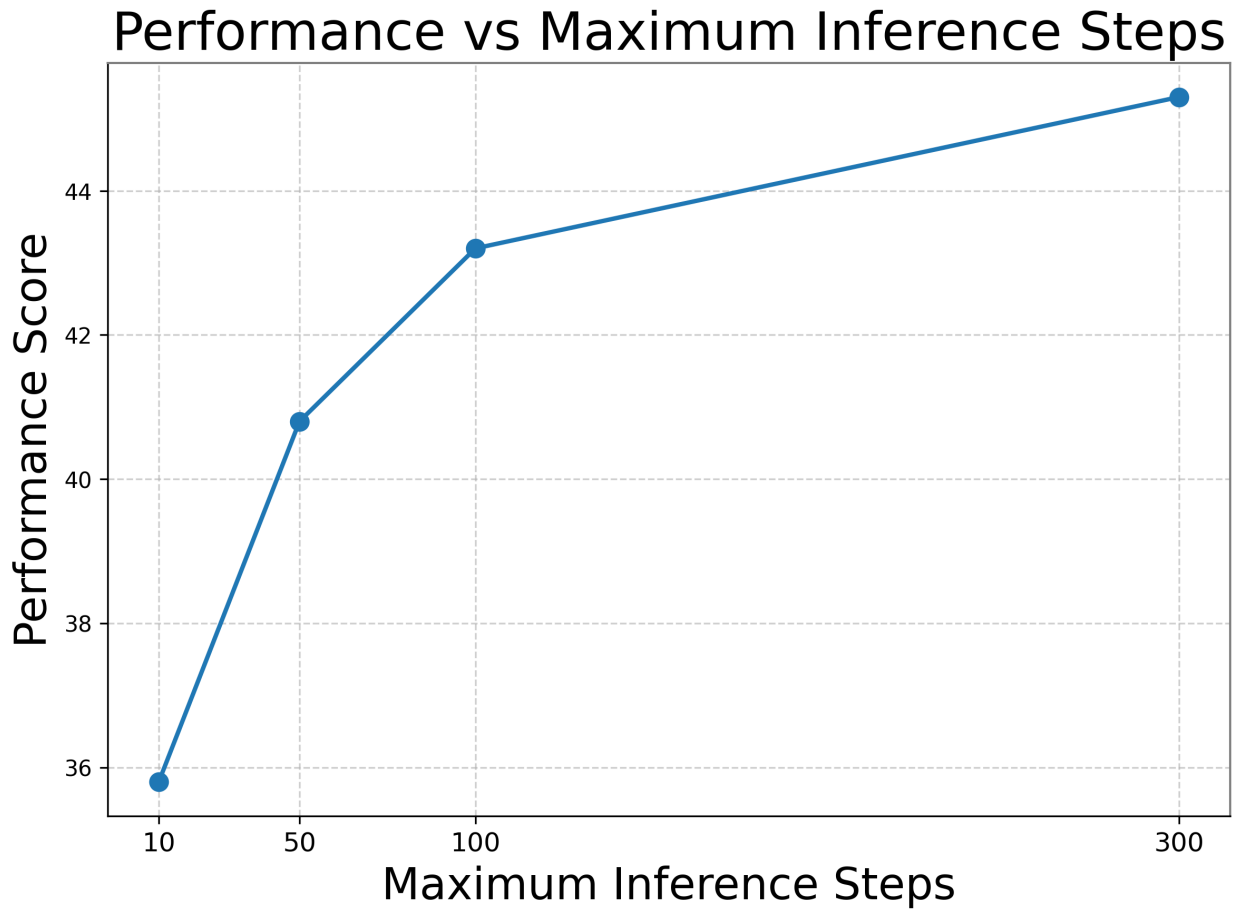



Figure 7.8: **Ablation study – inference steps**: higher limits yield further improvements, though the benefits gradually decay.

**Failure Case Analysis** According to our observations, we summarize the main causes of failures as follows: *(1) Element not visible:* In cases where the correct element is not visible on the current page (e.g., requires scrolling actions), zooming in on the current page is not beneficial. *(2) Non-interactable elements:* Some webpages contain extensive text that appears relevant but is not clickable, causing *RegionFocus* to zoom into these regions and subsequently fail to proceed due to the lack of interactable elements. For instance, on the Hugging Face homepage (https://huggingface.co/), there is abundant relevant text such as “Image-to-Text" and “Sentence Similarity,” which relates to agent tasks but is non-clickable. *(3) Wrong focal points:* In some rare cases, *RegionFocus* proposes the wrong focal point, directing attention to irrelevant regions. *(4) Action prediction failure:* Occasionally, despite identifying the correct focal point, all regional action predictions fail;

our observations suggest that employing SAM [14] could alleviate this issue as it further reduces the background clutter. *(5) Action aggregation failure:* Among the regional actions, the selected one may not be optimal. *(6) Others:* Other errors arise from general reasoning failures, bot detection, or reaching the maximum step limit.

## 7.5 Conclusion

We introduced *RegionFocus*, a visual test-time scaling approach that dynamically zooms in on relevant interface regions to address the clutter and ambiguity of modern GUIs. By integrating an *image-as-map* mechanism that marks key landmarks, our method provides transparent action records and improves coordinate-based action predictions. Experiments on Screenspot-pro and WebVoyager show substantial performance gains—even with a simple fixed-ratio bounding box strategy—highlighting the power of visual test-time scaling in enhancing interactive AI systems.

# CHAPTER 8

# Conclusion and Future Works

## 8.1 Conclusion

This dissertation argues that *structure* is a practical lever for building scalable and grounded visual intelligence. Visual data is generated by a structured world—compositional 3D objects and scenes projected into 2D, with viewpoints coupled by geometry. Visual tasks are also structured: correct behavior often hinges on a small, query-dependent subset of evidence. Finally, modern vision–language models (VLMs) are themselves structured systems: multimodal transformers route computation through token spans, attention heads, and specialized pathways. Across settings, treating data and model components as homogeneous can leave efficiency on the table and can encourage brittle behaviors such as shortcutting via language priors or exploiting background clutter.

The research chapters in this thesis develop structure-aware approaches across representations, data construction, objectives, and model adaptation. First, *Neural Shape Compiler* makes 3D structure explicit by translating between text, point clouds, and program-like shape descriptions via a unified discrete code, enabling cross-modal transfer and multi-level supervision. Second, *Cap3D* scales 3D supervision by generating multi-view captions from large 3D repositories, while *DiffuRank* further exploits multi-view structure by selecting informative views for supervision using generative-model signals. Third, *ViLP* exposes how visual–language shortcutting arises from language priors through controlled, prior-conflicting questions, and *Image-DPO* turns this diagnosis into a structured training signal that suppresses shortcut behavior via preference optimization.

Fourth, *Image-LoRA* leverages structure inside the model by restricting low-rank adaptation to the value pathway, the visual-token span, and a selected subset of attention heads, reducing adaptation cost while preserving text-only reasoning. Finally, *RegionFocus* uses structure at test time for interactive agents by dynamically zooming into task-relevant regions to reduce clutter and improve action grounding on screen-centric benchmarks.

Taken together, these results support a coherent thesis: explicitly leveraging structure—in the world, in the data, in learning signals, and in model internals—can simultaneously improve *scalability* (by focusing computation and supervision where it matters) and *grounding* (by discouraging reliance on spurious correlations and encouraging evidence-based decisions). Structural visual intelligence, in this sense, is not about imposing additional assumptions, but about exposing and exploiting the structure that is already present.

## 8.2 Future Works

The theme of this dissertation suggests several promising directions. I highlight a few that are especially aligned with the current trajectory of VLMs and with the broader goal of building reliable, general-purpose visual intelligence.

**Video as the next scaling frontier for visual structure.** A growing fraction of today's VLM capability comes from strong language modeling, with vision often functioning as a front-end that supplies a limited number of visual tokens. Video changes this regime: it can provide *trillions* of visual tokens, along with temporal continuity, motion cues, and repeated exposure to objects under changing viewpoints and lighting. Realizing this opportunity will require structure-aware processing to avoid naive, quadratic-cost scaling. Concretely, future work can explore hierarchical spatiotemporal tokenization, structured sampling over time (e.g., view/keyframe selection as a temporal analogue of DiffuRank), and dynamic region selection (a temporal extension of RegionFocus) that allocates compute to task-relevant events and entities rather than uniformly to all frames. More broadly, video offers an avenue to study how models build persistent object-centric representa-

tions and causal/world dynamics—forms of structure that remain underexploited in current VLM pipelines.

**Richer structure inside VLMs beyond "one transformer does everything."** This dissertation exploits internal structure via selective adaptation (Image-LoRA) and suggests that VLMs are not monolithic. A natural continuation is to study and design architectures where modularity is explicit and controllable: e.g., structured routing between perception and reasoning components, specialization of attention heads or blocks for grounding versus linguistic abstraction, and systematic mechanisms for updating only the components responsible for a target behavior. This includes understanding when and why visual grounding capabilities concentrate in a subset of heads or pathways, how such specialization evolves during training, and how to make it stable under continual learning. Ultimately, structure-aware modeling could enable VLMs that are both cheaper to adapt and more predictable in how they change.

**From text to code to math: compositional structure as a route to extrapolation.** Text has compositionality, but it is often permissive and underspecified. Code introduces stricter compositional and syntactic constraints. Mathematics pushes this further by demanding exactness and long-range logical consistency. A recurring limitation of transformer-based models is that they often excel at interpolation but struggle with extrapolation or robust systematic generalization. One direction is therefore to more tightly couple learning with domains that have strong, verifiable structure—including programming languages and formal mathematics. In particular, integrating training signals from proof assistants (e.g., Lean) could offer a path toward models that not only produce plausible answers but also produce *checkable* reasoning. This also opens a compelling multimodal angle: connect visual problems (geometry, diagrams, physical reasoning, 3D constraints) to formal statements and proofs, using structure as both supervision and evaluation.

**Structure-aware objectives and data construction for grounded alignment.** ViLP and Image-DPO show that diagnosing and correcting shortcutting benefits from *controlled structure* in the data and objectives. Future work can generalize this approach along several axes: (i) richer families

of counterfactual or prior-conflicting visual questions; (ii) preference data that targets grounding quality, not just answer correctness; (iii) objectives that jointly optimize for accuracy, calibration, and evidence attribution; and (iv) interactive feedback loops where the model collects information (e.g., by zooming, querying, or taking actions) and is trained to justify decisions based on acquired evidence. Beyond benchmarking, the broader goal is to make grounding failures *legible* and therefore correctable.

**Unifying structured representations across 3D, VLMs, and agents.** The early chapters show the value of structured intermediate representations (e.g., discrete 3D codes and program-like descriptions). A longer-term direction is to unify these with modern VLM and agent pipelines: using structured latent variables (objects, parts, relations, programs) not only as outputs, but as internal "working memories" that mediate between perception and action. This could support systems that plan over structured states, generalize across domains, and reuse learned skills. It also raises practical questions about how to interface discrete structure with continuous perception, and how to learn structured states from weak supervision at scale (especially from video and interaction traces).

**Closing perspective.** Across these directions, the guiding hypothesis remains the same: the most scalable path to grounded visual intelligence is not to treat all data and all model components uniformly, but to identify the structure already present and to align representations, learning signals, computation, and evaluation with it. This dissertation provides several concrete instances of that principle; extending them to video, formal reasoning, and richer model modularity is a natural next step toward more robust and general visual intelligence.

# APPENDICES

# APPENDIX A

# Supplementary Material for Neural Shape Compiler

## A.1 Data Collection

### A.1.1 3D Shape Assets

As mentioned in Section 4, we collect a total of 140,419 shapes from ShapeNet (*Chang et al.*, 2015), ABO dataset (*Collins et al.*, 2021), and Shape Program (*Tian et al.*, 2019). For ShapeNet, we use the most recent release version (ShapeNetCore.v2), which contains 52,472 shapes from 55 categories. ABO dataset contains 7,947 shapes from 98 categories and has nine overlapped shape categories with ShapeNet. We adopt the 4 Chair and 10 Table templates in Shape Program (*Tian et al.*, 2019) to synthesize 40,000 shapes in each category. We do not distinguish shape categories during training for both *Point*VQVAE and ShapeCode Transformer. We will normalize the input point clouds into a unit ball for addressing dataset gaps.

### A.1.2 Shape-Text Pairs

We collect 107,371 (*Point Cloud*, *Structure-Related Text*) pairs by adjusting and annotate the current datasets, including Text2Shape (*Chen et al.*, 2018), ShapeGlot (*Achlioptas et al.*, 2019a), and ABO (*Collins et al.*, 2021) datasets. We take tables from Text2Shape, chairs from ShapeGlot,

and all shapes from the ABO dataset. Therefore, our dataset consists of furniture, most of which are tables and chairs. We list our considerations and data collection details below.

**Text2Shape:** (*Chen et al.*, 2018) provides descriptions for both chairs and tables. Through careful inspection, we found there are a large number of descriptions regarding artificial colors (e.g., red, blue), textures (e.g., green grids on the top), and materials (e.g., wooden, steel). Since this work aims to focus on the 3D geometry and will not predict any colors for output point clouds, we delete those artificial texture contents but remain the geometrical ones. Also, compared to Text2Shape, ShapeGlot (*Achlioptas et al.*, 2019a) provides much more geometrical descriptions for chairs, where the chairs have a very high overlap with chairs in Text2Shape. We thus discard the chair descriptions and only adopt the table descriptions of Text2Shape in our data. For a few data which lack structural descriptions, we will annotate them manually. Some examples are in Figure A.1.

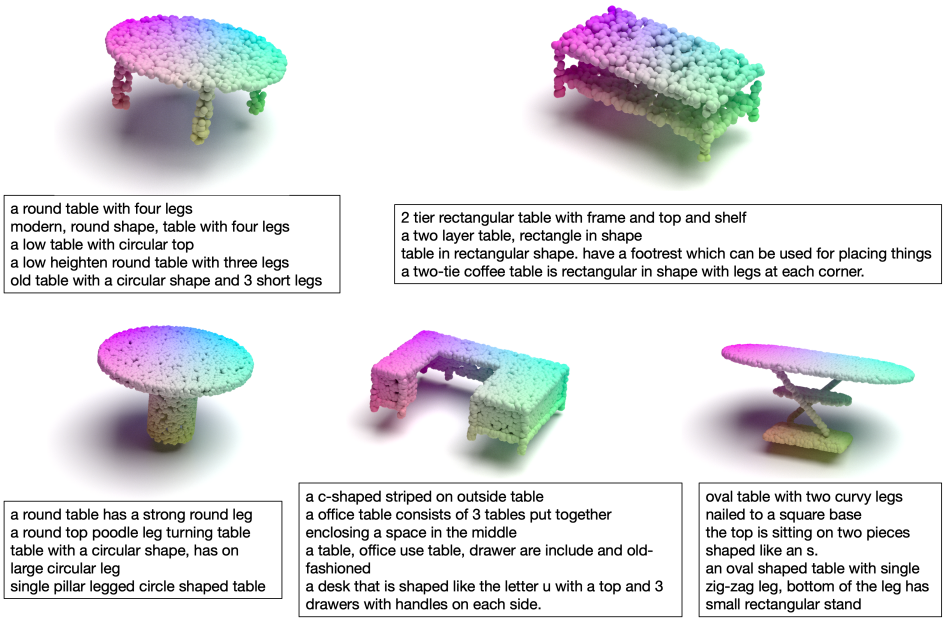



Figure A.1: Random examples in adjusted Text2Shape dataset. One description per sentence.

**ShapeGlot:** (*Achlioptas et al.*, 2019a) provides many meaningful structural descriptions for chairs. However, the descriptions are about triple relationships. It provided users with two shapes and a comparative description and asked users to choose which shape most satisfied the description, i.e. (*Shape A*, *Shape B*, *Text*). We transfer the triples into doubles by assigning the text to the target shape according to annotations. However, some texts lose their meanings if they do not have the

control shape. Furthermore, we found single text in ShapeGlot is usually not very informative that can plot the target shape accurately. In the end, we filter out the less-meaningful texts and concatenate several shapes' texts together as one description. Also, we remove superlative words, like longest, because they usually cannot hold across the entire dataset. For a few data which lack structural descriptions, we will annotate them manually. Some examples are in Figure A.2.

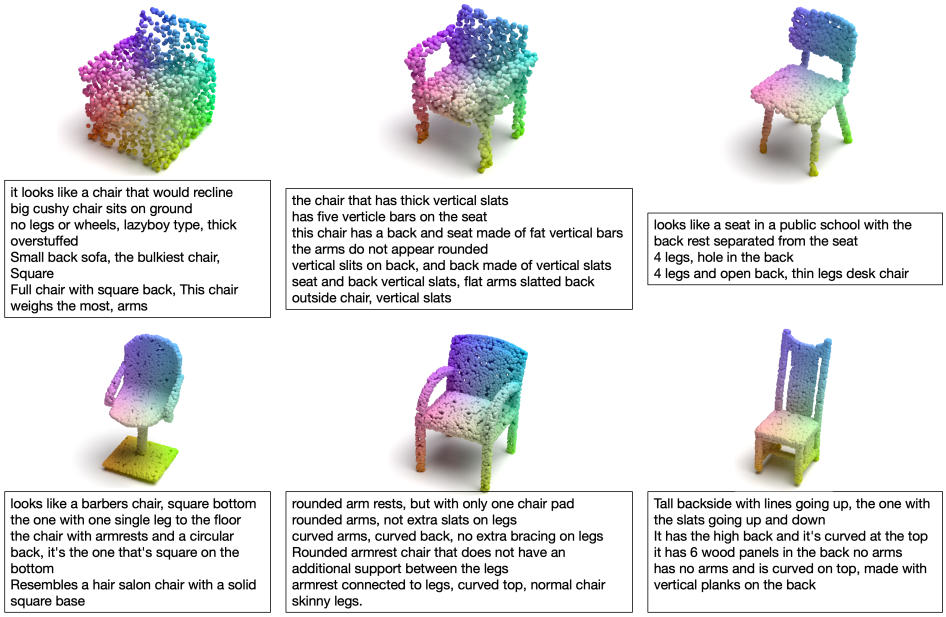



Figure A.2: Random examples in adjusted ShapeGlot dataset. One description per sentence.

**ABO:** (*Collins et al.*, 2021) provides high-quality CAD models and descriptions in different languages for amazon products. We only use the description of the 'item_name' under the language tag 'en_US'. To add structure-related texts to ABO shapes, we collect more descriptions by manually annotating. Also, since ABO consists of amazon products, some original descriptions contain brand names, such as "amazon basics" and "ravenna", which are not connected to shape geometry. Also, some descriptions contain product dimensions like $16.0 \times 8.2 \times 7.4$. Since we normalize all point clouds into a unit ball, those dimensions are not connected to its scale. Therefore, we deleted the descriptions related to brand names and dimensions in the descriptions. Some examples are in Figure A.3.

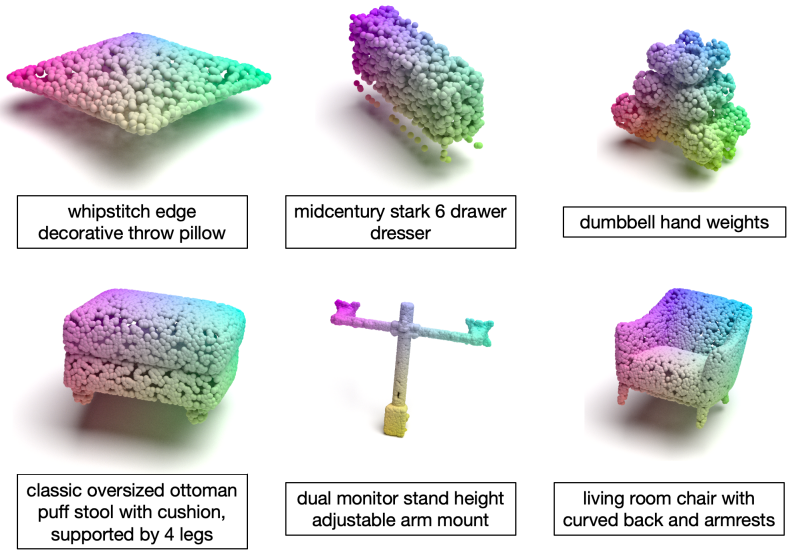



Figure A.3: Random examples in adjusted ABO dataset. One description per sentence.

### A.1.3 ParitalShape-CompleteShape Pairs

We used the data from GenRe (*Zhang et al.*, 2018) which has 20 random view renderings of airplanes, cars, and chairs from ShapeNet, for a total of 20 $\times$ 9,646 training (*point-clouds*, *point-clouds*) pairs and 20 $\times$ 1,382 test (*point-clouds*, *point-clouds*) pairs. We sample 200 points from the above depth images as inputs, sample 2048 points from the corresponding 3D shapes as targets, and evaluate over all the provided 20 views. Unlike previous benchmarks (*Zhou et al.*, 2021), which train a model for a single shape category, we train every completion model with all training data.

## A.2 Training & Model Details

This section provides the training and model details for our *Point*VQVAE and ShapeCode Transformer used in our main paper, and the *Program*VQVAE we mentioned in Section 2.3.1. We implement and train our models with PyTorch (*Paszke et al.*, 2019) and 8 A40 GPUs.

### A.2.1 *Point*VQVAE

For *Point*VQVAE, we adopt the cosine annealing strategy of SGDR (*Loshchilov and Hutter*, 2016) (w/o restarts) as our optimization scheduler and set the initial learning rate 0.002 with $8 \times 32$ batch size and 200 epochs.

*Point*VQVAE encoder has two hierarchical layers ($\#T = 2$). In the first layer, we sample 512 center points. Then, we gather information from 64 points within $R_{T_1} = 0.1$ ball around each center point with two fully connected layers [64, 512] and [512, 512]. After the first layer, our point cloud size will be reduced to 512 from 10,000 (original input size). Similarly, in the second layer, we sample 128 center points, and do $R_{T_2} = 0.4$ ball query with 512 points. We also adopt two fully connected layers here with shapes [512, 512]. After the encoding phase, we will have 128 embeddings and look for the closest embedding in the codebook that has the size of [512, 512]. We concatenated the indexed embedding to form a [128, 512] embedding and sent it to the decoder. The structure of *Point*VQVAE decoder is shown in Figure 2, we have batch norm 1D (*Ioffe and Szegedy*, 2015) and ReLU (*He et al.*, 2015) following each convolutional layer. The output of the residual-like structure (Figure 2) has the shape [#Batch Size, 512, 128]. We then applied a max-pool layer to the last dim of the output to obtain an embedding with shape [#Batch Size, 512, 1]. The max-pool layer can eliminate the effect caused by the concatenation order of the codebook embedding, similar to the use of max-pool layers in PointNet (*Qi et al.*, 2017a). After the max-pool layer, we finally have two Convolutional 1D layers with [512, 1024] and [1024, 2048 * 3] to predict 2,048 points. Batch norm 1D and ReLU are also used between the last two layers.

We use Chamfer Distance (CD) and Earth Mover's Distance (EMD) as the reconstruction loss to serve as one of the objectives of *Point*VQVAE. Specifically, we adopt the implementation of CD in Pytorch3D (*Ravi et al.*, 2020) and implemented an EMD approximation via auction algorithm (*Bertsekas and Castanon*, 1989) with the help of codes provided in (*Fan et al.*, 2017; *Liu et al.*, 2020b). The EMD codes will be released with a detailed report.

$$\mathscr{D}_{EMD}(P_1,P_2) = \min_{\phi:P_1\to P_2} \sum_{x\in P_1} \|x-\phi(x)\|_2, \;\; \phi \text{ is a bijection} \tag{A.1}$$

$$\mathscr{D}_{CD}(P_1,P_2) = \sum_{x\in P_1} \min_{y\in P_2} \|x-y\|_2^2 + \sum_{y\in P_2} \min_{x\in P_1} \|x-y\|_2^2 \tag{A.2}$$

$$\tag{A.3}$$

The overall objective for optimizing *Point*VQVAE adopted the one used in (*Van Den Oord et al.*, 2017) and is shown below, where sg represents stop gradient operations, $\mathscr{Z}_e(p)$ refers to the output of our encoder, $e$ is the nearest embedding in our codebook with the encoder output. The first two terms are the reconstruction loss to push the final output of *Point*VQVAE closer to input point clouds. The third term is $\mathscr{L}_2$ loss to move the embedding in our codebook closer to the corresponding encoder output. The fourth one is the commitment loss to constrain those embedding growing.

$$\mathscr{L}_{total} = \mathscr{D}_{EMD} + \mathscr{D}_{CD} + \|\text{sg}[\mathscr{Z}_e(p)] - e\|_2^2 + \|\mathscr{Z}_e(p) - \text{sg}[e]\|_2^2$$

### A.2.2 ShapeCode Transformer

We also adopt the cosine annealing strategy as our optimization scheduler and set the initial learning rate 0.001 with $8 \times 24$ batch size and 100 epochs. For generality, we adopt the simplest Transformer (*Vaswani et al.*, 2017) with depth 5, wide 64, and 8 attention heads. We integrate full attention masks (*Child et al.*, 2019) to force it to do autoregressively predictions (*Van den Oord et al.*, 2016).

As mentioned in Section 2.3.2, we have different positional embedding for each type of data pair (i.e., task), which will be optimized during the training. Since we pad 0 at the leftmost of our data, the positional embedding for the conditional code will have an extra token. For example, if we conduct (*Text*, *Point Cloud*) transformation, we will have $P_{text} = 256+1 = 257$ positional tokens for Text and $P_{point} = 128$ for Point Cloud. Therefore, we have a total of $257+128 = 385$ positional

tokens for (*Text*, *Point Cloud*). For each positional embedding, we have the embedding dimensional 512. As a result, we have embedding size $[385, 512]$ for the task (*Text*, *Point Cloud*). For each task, Neural Shape Compiler has different positional embeddings but the same process described above.

Also, ShapeCode Transformer has a self-maintained embedding matrix for each type of code, as illustrated in Section 2.3.2. We also use (*Text*, *Point Cloud*) transformation as example, where Text code has $N_{text} = 256$ tokens and Point Cloud code has $N_{point} = 128$. The embedding dimension is 512. Therefore, we have embedding matrix with size [$N_{text}$, 512], [$N_{point}$, 512], and [$N_{program}$, 512] for Text, Point Cloud, and Program, respectively. Regarding token length and vocabulary size for each shape abstraction, $\mathscr{C}^{point}$ has $N_{point} = 128$ tokens with a vocabulary size of 512 via argmaxing from the *Point*VQVAE encoder output; $\mathscr{C}^{text}$ has $N_{text} = 256$ tokens of vocabulary size $49,408$, we pad between the last valid text token and the beginning of another token with a value between $[49408, 49408 + 256]$; $\mathscr{C}^{program}$ has $N_{program} = 240$ tokens and a vocabulary size of 78.

The overall objective of ShapeCode Transformer is the summation of loss computing over each pair of data: (1) (*Text*, *Point Cloud*); (2) (*Point Cloud*, *Text*); (3) (*Point Cloud*, *ShapeProgram*); (4) (*Partial PointCloud*, *Complete PointCloud*). For each pair, we define the loss over every token with Cross-Entropy loss as shown below, where $\mathscr{L}$ indicates cross-entropy. Noting that, for program codes $\mathscr{C}^{program}$, we will turn negative integers into positive integers, like mapping [-20, 20] to [0, 40].

$$\sum_{k} \mathscr{L}(c_k, c_k^{gt} | [0, \cdots, c_{k-1}])$$

### A.2.3 *Program*VQVAE

Currently, Shape Programs (*Tian et al.*, 2019) only has limited discrete parameters, and thus, we adopted the way of discretizing parameters into codes in our main paper. For example, we will map integers in [-20, 20] to [0, 40] to assign unique positive integers for each parameter. Although the current shape program can already cover a lot of 3D shapes, it may be upgraded to encode more detailed structures of 3D shapes that need to involve continuous parameters. For example, the

statement *draw('Top', 'Square', P=(-1,0,0), G=(2,5))* in Figure 1 may have decimal parameters *draw('Top', 'Square', P=(-1.2,0,0.3), G=(2,5.7))*. Furthermore, the primitive basis used in the shape program may be replaced with a functional basis (*Mescheder et al.*, 2019; *Genova et al.*, 2020). As mentioned in Section 3.1, we also designed *Program*VQVAE to transform shape programs into codes to fill the gap in handling continuous parameters.

The design of *Program*VQVAE is pretty similar to our *Point*VQVAE, where we leverage the most basic concepts of vector quantization and adopt the straight-through gradient estimator and the vanilla codebook objective used in (*Van Den Oord et al.*, 2017) for optimizing. Given a statement, we will turn its statement type into a one-hot vector and concatenate it with its parameters as the input to *Program*VQVAE. For example, we will turn *draw('Top', 'Square', P=(-1,0,0), G=(2,5))* into $[3,-1,0,0,2,5,0,0]$ where the value 3 is the represents the statement type 'draw'. The encoder and decoder will be two vanilla Transformer (*Vaswani et al.*, 2017) with depth 3 and MLP dimension 32, and the codebook is [512, 512], the same size as we used in *Point*VQVAE. We use cross-entropy to compute the loss for predicting statement types and $\mathscr{L}_2$ loss for regressing statement parameters.

We synthesize 200,000 (*Point Cloud*, *Program*) pairs for training the *Program*VQVAE. On 10,000 validation data, it achieved 100% accuracy in predicting statement types and < 0.01 L2 distance in parameter regression. The results show the proposed *Program*VQVAE can well learn the distribution of our synthesized shape programs.

### A.2.4 Baselines Implementation Details

This paper compared various baselines across different tasks to examine the performance of the proposed framework. We generally follow their provided codes and default configurations for each baseline, including training epochs, learning rates, and other hyper-parameters. We only modify the code when changes are needed to use our dataset. Some modifications are listed in this section.

- Shape IMLE (*Liu et al.*, 2022b) preprocesses all texts with Bert (*Devlin et al.*, 2018) in its framework. We adopted the same way and used *BertTokenizer* of the pre-trained flag "bert-base-uncased" to process all the texts in our dataset.

- We adopted *clip.tokenize* and *clip.encode_text* to encode all test texts and input them into CLIP-Forge (*Sanghi et al.*, 2022) for quantitative comparisons. We input the text prompt to DreamField (*Jain et al.*, 2021) by the provided command line in Gihub of *Jain et al.* (2021) for qualitative comparisons. In our environment, we need 4 GPUs of 2048 Gb memory to run a single text input with DreamField (*Jain et al.*, 2021). It takes more than eight days to finish the default 10,000 iterations for single text input. Therefore, it is impractical to test DreamField in our test set, which consists of 15,000 text-shape pairs.

- CWGAN (*Chen et al.*, 2018) provided processed data as templates. We follow their data formats to process our data and update each entry, including *caption_tuples*, *caption_matches*, *vocab_size*, *max_caption_length*, and *dataset_size*. We adopted spaCy (*Honnibal et al.*, 2020) to tokenize texts as mentioned in *Chen et al.* (2018). (*Chen et al.*, 2018) simply set the token number in a linear increase rule based on their data order, while we set our token number based on our data order. We also disregarded descriptions with more than 96 tokens as (*Chen et al.*, 2018) did.

- For AutoSDF (*Mittal et al.*, 2022), their current codebase did not support training generation models conditioned in texts; we do not compare it for now. We will try to add their results once the code is updated.

- For the point cloud completion baseline PVD (*Zhou et al.*, 2021), we jointly train their models with three data classes. This differs from their original paper, which trains separate models for each shape class.

## A.3 *Point*VQVAE Discussions

### A.3.1 Analysis of Learned Codebook

This section provides some preliminary analysis of the learned codebook of *Point*VQVAE. The codebook is the interface to connect the output of *Point*VQVAE encoder and the input of

*Point*VQVAE decoder. In our experiments, the codebook size is [512, 512], and we have 128 embedding for a single input point cloud.

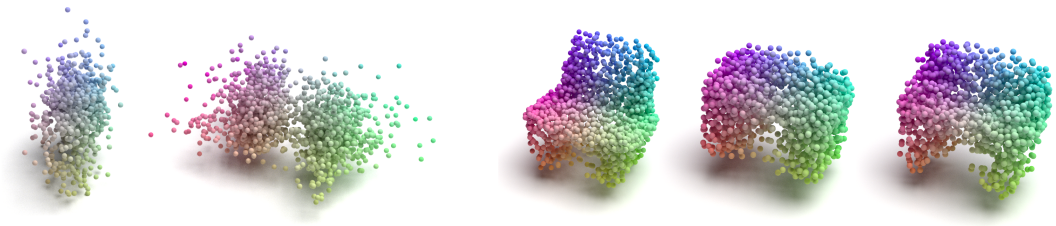


Figure A.4: The right three shapes are reconstruction results of random codes and the left two shapes are reconstruction results of indexed codes illustrated in the text.

For each input point cloud, we will have 128 codes to index the embedding and feed it into the decoder later. We look at how many different embeddings of the codebook will be used for a single input point cloud. We compute the number with whole ShapeNet (*Chang et al.*, 2015) shapes, where we auto-encode shape and compute the average ratio of unique codes. The result is 44.28, which indicates point cloud code $\mathscr{C}^{point}$ will have multiple replicate code numbers. Does this indicate that the codebook learns the basic primitives (*Deprelle et al.*, 2019), selects the suitable ones for a specific point cloud, then fuses them for reconstruction? If so, will those primitives be perceptually understandable to our humans? To study this, we design indexed codes with all one number $[x, x, \cdots, x]$, like $[1, 1, \cdots, 1]$, length = 128. Those codes visualize the codebook's #$x$ embedding. We feed the special codes into our decoder to reconstruct the corresponding embedding and show two examples at the left of Figure A.4. Results show the embedding is pretty random to humans and may exist in certain symmetry structures. Since the reconstruction results are highly correlated to the decoder weights, the way we adopted above may not be the ideal solution. We will try to develop other methods to understand how the 3D codebook works in the future. Besides, we also tried to randomly generate the codes and reconstruct them to see if the random samples from the latent discrete space are meaningful. Results are shown in the right of Figure A.4, where they are somewhat reasonable in the structure. We also show an example in Figure A.5 where we interpolate two different shape codes and reconstruct the intermediate results.

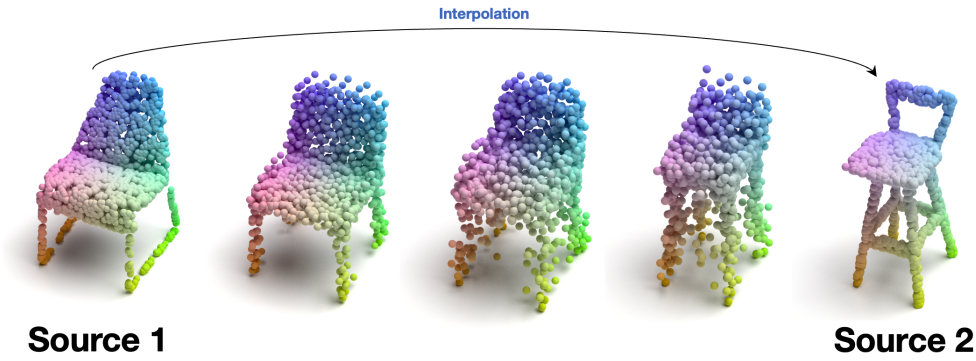



Figure A.5: A random example of interpolating latent codes between two resource shapes of our proposed *Point*VQVAE.

### A.3.2 Empirical Studies

In this section, we conduct empirical study about *Point*VQVAE, include ablation studies and comparisons. We changed the number of points in the last layer of the encoder (the color points in Figure 2.3) from 1 to 128. When the point number equals 1, our encoder equals the standard encoder of PointNet++ (*Qi et al.*, 2017b) which uses a global pooling layer. Table A.1 shows such a design brings a catastrophic performance drop due to it being significantly difficult for the codebook to encode holistic point clouds instead of parts of the point clouds.

| Variants | CD ↓ |
|---|---|
| *Point*VQVAE globalpool | 16.63 |
| *Point*VQVAE final-16 | 10.26 |
| *Point*VQVAE final-32 | 8.57 |
| *Point*VQVAE final-64 | 7.34 |
| *Point*VQVAE final-128 | 6.93 |
| *Point*VQVAE final-256 | 6.78 |

| Methods | CD ↓ |
|---|---|
| l-GAN (*Achlioptas et al.*, 2018) | 9.23 |
| Point Flow (*Yang et al.*, 2019) | 7.76 |
| ShapeGF (*Cai et al.*, 2020) | 5.97 |
| DiffGen (*Luo and Hu*, 2021) | 5.62 |
| 3DCM (*Cheng et al.*, 2022) | 6.32 |
| *Point*VQVAE Limited | 6.93 |
| *Point*VQVAE | 5.98 |
| Oracle | 3.09 |

Table A.1: Point Cloud auto-encoding performance on ShapeNet. CD is multiplied by $10^3$. X in *Point*VQVAE final-X means how many points we sampled in the last layer of the encoder. Globalpool indicates only having one point with the global receptive field. All the models besides *Point*VQVAE are trained on ShapeNet (*Chang et al.*, 2015) shapes. *Point*VQVAE is trained on all our 3D shapes.

Also, we compare with some point clouds based auto-encoders (*Achlioptas et al.*, 2018; *Yang*

*et al.*, 2019; *Luo and Hu*, 2021; *Cheng et al.*, 2022). For each shape, we sample 2048 points thereon and use Chamfer Distance (CD) to measure the difference between inputs and reconstruction results. We will report Chamfer Distance $\mathscr{D}_{CD}(P_1, P_2)$ as the summation of both two-direction $P_1 -> P_2$ and $P_2 -> P_1$. EMD is not reported because each method has its implementation of EMD, and none of the methods can guarantee an approximation of EMD based on their GitHub code. In contrast, all the used CD implementations are correct. The results are shown in Table A.1. The numbers under "Oracle" mean the lower bound of reconstruction, obtained by computing the distance between two different samplings from the same shape meshes. Compared to *Point*VQVAE Limited, *Point*VQVAE achieved higher scores which means training over more 3D data can help *Point*VQVAE model the 3D shape distribution and also not reduce its performance on the part of the training data.

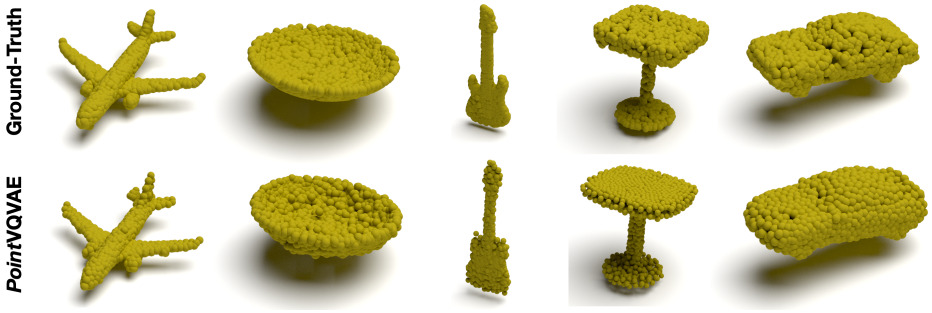



Figure A.6: Random reconstruction examples of *Point*VQVAE.

## A.4 *Point Cloud* $\Longrightarrow$ *Program* Discussions

In this section, we mainly discuss two points around *Point Cloud* $\Longrightarrow$ *Program*: (1) the generated sample numbers used in our shape program experiments; (2) the guided adaptation used in (*Tian et al.*, 2019).

We adopt 48 as the sample number in *Point Cloud* $\Longrightarrow$ *Program* experiments because all other tasks adopt batch size 48. Here, we provide a more thorough study of the sample numbers. We test Shape Compiler performance in all six shape categories with sample numbers [1, 2, 4, 8, 16, 48, 96, 128, 256]. The results are shown in Figure A.7. The results show that the proposed method achieved higher performance by increasing the generated sample numbers. This is also one of the

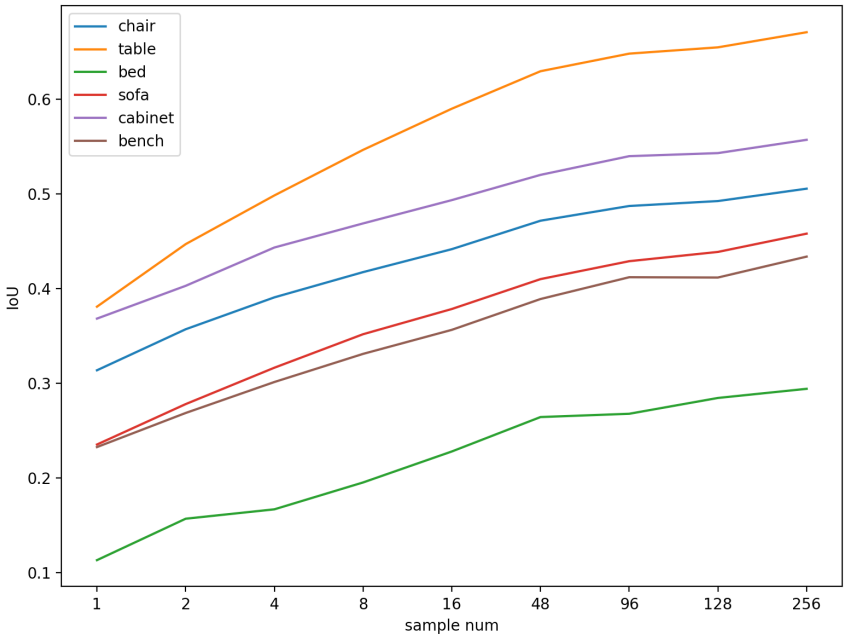



Figure A.7: *Point Cloud* $\Longrightarrow$ *Program* performance versus the shape generated sample numbers.

benefits of the generation framework we adopted in our approach.

Shape Program (*Tian et al.*, 2019) used guided adaptation in finetuning their models to new test data. They achieved higher performance by using guided adaptation. However, in our experiments, we intended not to follow (*Tian et al.*, 2019) to perform guided adaptation. We choose this way because we aim to take a step further in shaping program generation for practical uses. In real-world applications, we usually cannot afford the extra training time for conducting "guided adaptation"; sometimes, we even do not have enough computation to run adaptation (e.g., edge computing). Also, generating shape programs without "guided adaptation" can enable us to use one class-agnostic model to process shapes from different categories instead of having class-specific models for each class. The proposed framework, Shape Compiler, only has one class-agnostic model for processing six different shape categories. However, *Tian et al.* (2019) will need to train six different models to process the test shape if they use guided adaptation. Combining the advantages of time-saving and class-agnostic, PointCloud-to-Program could take a step toward achieving real-world impact.

## A.5 More Results

In this section, we provide more generated samples by the proposed Neural Shape Compiler. Extra results are contained in the supplementary.

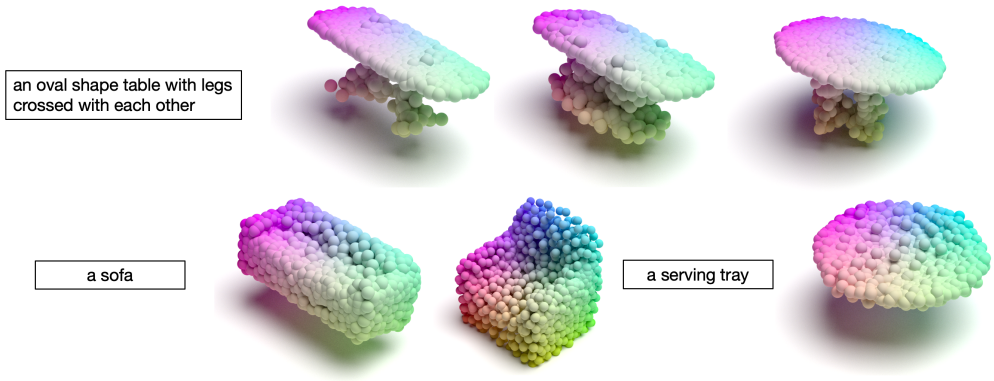



Figure A.8: Generated shapes by Neural Shape Compiler given different text prompts. Shape Compiler can generate multiple shapes given a text query.

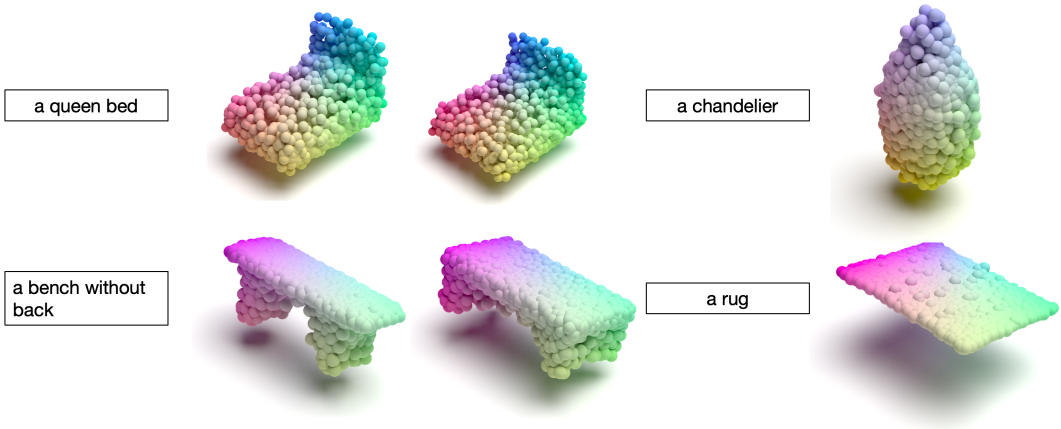



Figure A.9: Generated shapes by Neural Shape Compiler given different text prompts. Shape Compiler can generate multiple shapes given a text query.

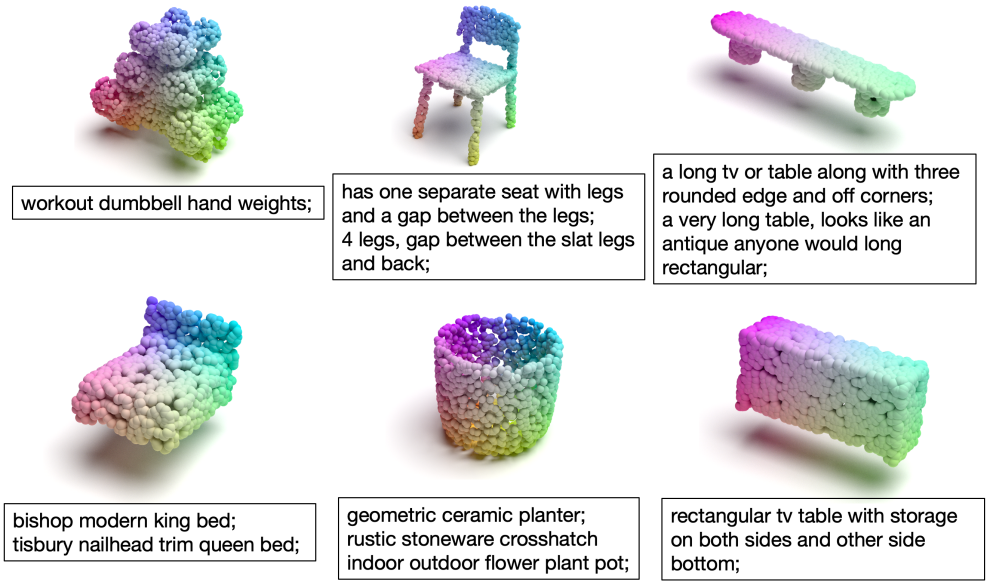



Figure A.10: *Point Cloud* $\Longrightarrow$ *Text* results by Neural Shape Compiler given different point clouds. Shape Compiler can generate multiple captions given a point cloud query. One description per sentence.

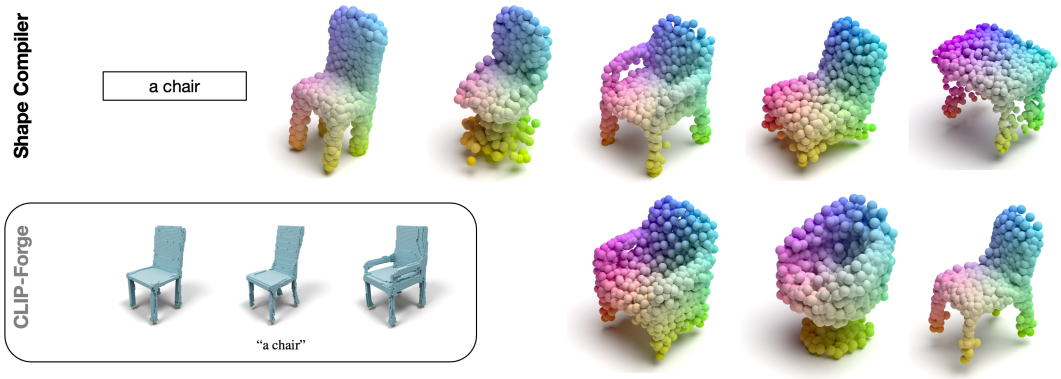



Figure A.11: Directly comparing the performance of Neural Shape Compiler with CLIP-Forge (*Sanghi et al.*, 2022) results on a simple text prompt.

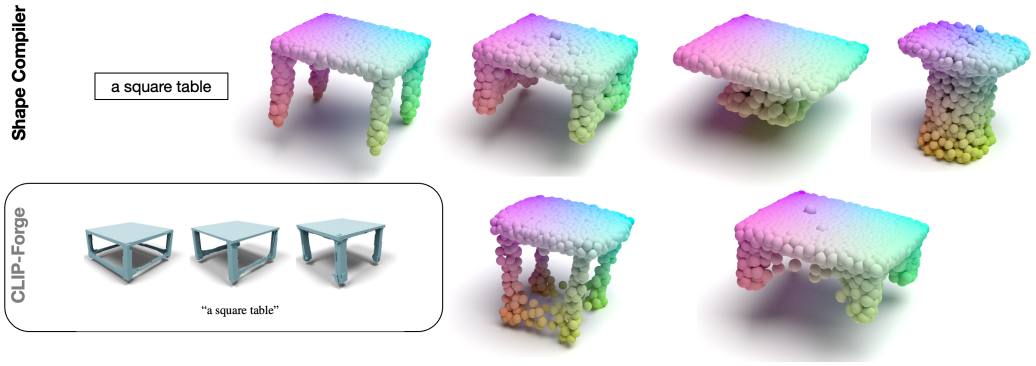



Figure A.12: Directly comparing the performance of Neural Shape Compiler with CLIP-Forge (*Sanghi et al.*, 2022) results on a simple text prompt.

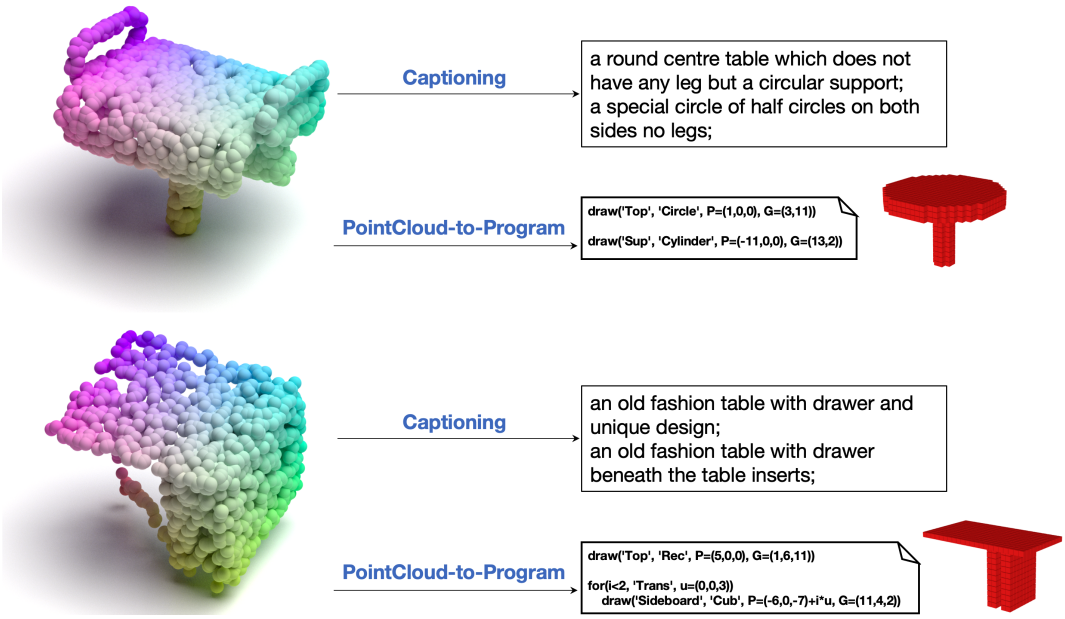



Figure A.13: *Point Cloud* $\Longrightarrow$ *Text* and *Point Cloud* $\Longrightarrow$ *Program* results by Neural Shape Compiler given the input point clouds. One description per sentence.