Experiencing InstructPipe: Building Multi-modal AI Pipelines via Prompting LLMs and Visual Programming

Zhongyi Zhou[†‡], Jing Jin[‡], Vrushank Phadnis[‡], Xiuxiu Yuan[‡], Jun Jiang[‡], Xun Qian[‡], Jingtao Zhou[‡], Yiyi Huang[‡], Zheng Xu[‡], Yinda Zhang[‡], Kristen Wright[‡], Jason Mayes[‡], Mark Sherwood[‡], Johnny Lee[‡], Alex Olwal[‡], David Kim[‡], Ram Iyengar[‡], Na Li[‡], Ruofei Du[‡*]
1: The University of Tokyo, Japan
2: Google Research, USA

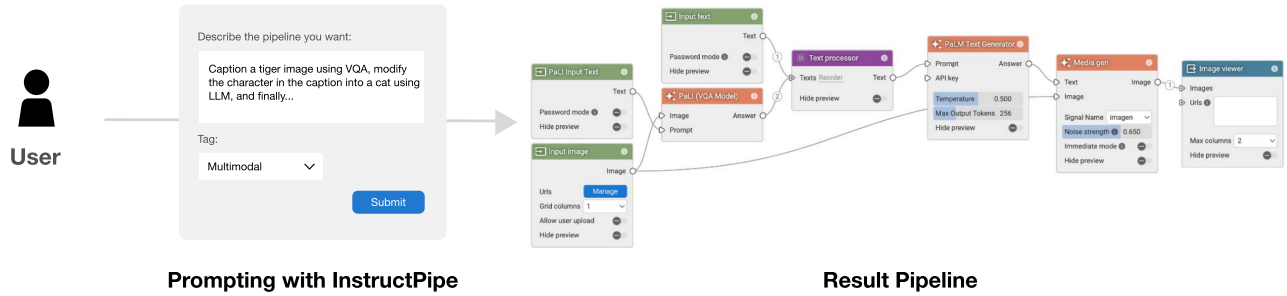


Figure 1: We demonstrate InstructPipe, an AI assistant that enables users to start prototyping machine learning (ML) pipelines with text instructions. To achieve this, InstructPipe leverages two LLM modules and a code interpreter to generate pseudocode of a target pipeline, and the interpreter renders a pipeline in the node-graph editor for further human-AI collaboration.

ABSTRACT

Foundational multi-modal models have democratized AI access, yet the construction of complex, customizable machine learning pipelines by novice users remains a grand challenge. This paper demonstrates a visual programming system that allows novices to rapidly prototype multimodal AI pipelines. We first conducted a formative study with 58 contributors and collected 236 proposals of multimodal AI pipelines that served various practical needs. We then distilled our findings into a design matrix of primitive nodes for prototyping multimodal AI visual programming pipelines, and implemented a system with 65 nodes. To support users' rapid prototyping experience, we built InstructPipe, an AI assistant based on large language models (LLMs) that allows users to generate a pipeline by writing text-based instructions. We believe InstructPipe enhances novice users onboarding experience of visual programming and the controllability of LLMs by offering non-experts a platform to easily update the generation.

CCS CONCEPTS

• **Computing methodologies** → Visual analytics; *Machine learning*; • **Software and its engineering** → **Visual languages**.

KEYWORDS

Visual Programming; Large Language Models; Visual Prototyping; Node-graph Editor; Graph Compiler; Low-code Development; Deep Neural Networks; Deep Learning; Visual Analytics

ACM Reference Format:
Zhongyi Zhou[†‡], Jing Jin[‡], Vrushank Phadnis[‡], Xiuxiu Yuan[‡], Jun Jiang[‡], Xun Qian[‡],, Jingtao Zhou[‡], Yiyi Huang[‡], Zheng Xu[‡], Yinda Zhang[‡], Kristen Wright[‡], Jason Mayes[‡],, Mark Sherwood[‡], Johnny Lee[‡], Alex Olwal[‡], David Kim[‡], Ram Iyengar[‡], Na Li[‡], Ruofei Du[‡]. 2024. Experiencing InstructPipe: Building Multi-modal AI Pipelines via Prompting LLMs and Visual Programming. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '24), May 11–16, 2024, Honolulu, HI, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3613905.3648656

1 INTRODUCTION

A visual programming interface provides users with a workspace to program in a node-graph editor. This format allows for the creation of a pipeline using visual elements like nodes and connections, enhancing the ability to prototype various applications without deep knowledge of programming languages. The field has seen increased interest due to advancements in machine learning (ML). Open-source ML libraries, *e.g.*, TensorFlow [1], PyTorch [11], and Hugging Face [15], offer a wide range of pre-built modules, expediting AI project development and experimentation.

*Corresponding author: me [at] duruofei [dot] com; the first author conducted this work during Google internship, also contact: zhongyi.zhou.work [at] gmail [dot] com

CHI EA '24, May 11–16, 2024, Honolulu, HI, USA

ACM ISBN 979-8-4007-0331-7/24/05
https://doi.org/10.1145/3613905.3648656

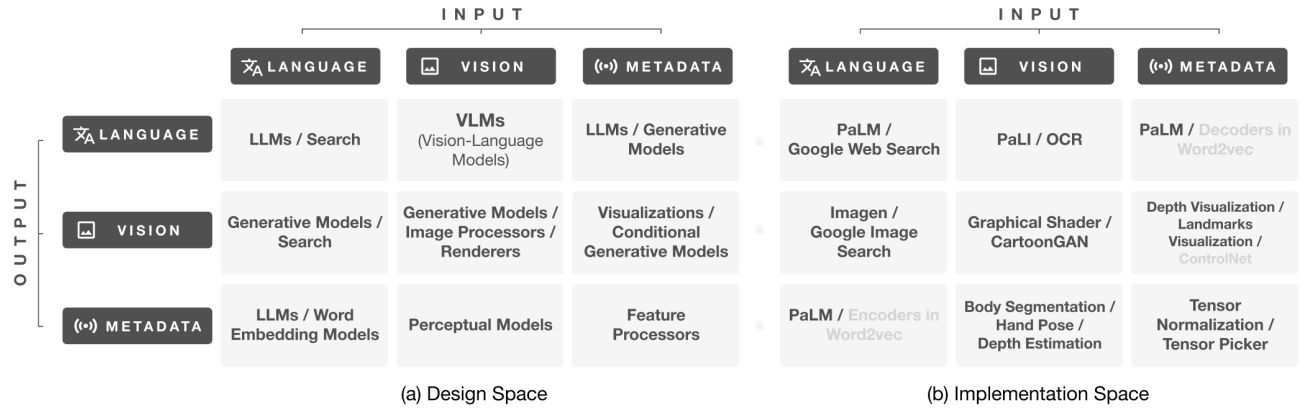


Figure 2: Design and implementation spaces of nodes for visual prototyping. (a) sketches a high-level overview of proposed nodes in a matrix and (b) lists our implemented nodes (in black) and potential nodes (in gray).

Visual programming offers the community a platform for users to interactively explore the creative use of existing ML models by allowing them to freely select and connect nodes into a new functional pipeline.

However, this streamlined approach has its limitations. The utility of a visual programming platform relies heavily on its predefined nodes — users' creative process will be disrupted even with one missing node. Compared to various built-in functions in the programming language, visual programming typically provides a small number of pre-defined nodes (*e.g.*, 38 nodes in Visual Blocks [6]). The community addresses this issue in a *reactive* development cycle by collecting feature requests from users and then assigning tasks to developers. We believe that the lack of community guidelines makes it challenging to address this issue in a *proactive* manner.

With design guidance, as developers continue to add nodes for enhanced system utility, it will eventually clutter the node library, causing a high cognitive burden to perceive them. Even worse, users typically build a pipeline *"from scratch"*, *i.e.*, selecting nodes, ideating the pipeline structure and finally connecting nodes from a completely empty workspace. A large pool of nodes can easily overwhelm users in this creative process, causing an unsatisfactory user experience (*i.e.*, the usability issue). Similar issues also exist when users write code using programming languages (with many built-in functions and libraries), but recent advances in Large Language Models (LLMs) show that such challenges can be addressed. For example, GitHub Copilot [8] makes it possible to generate code by simply describing users' requirements in natural language. Therefore, we hypothesized that similar AI assistants could also benefit visual programming interfaces by reducing users' workload in their node editing experiences.

In this paper, we present 1) a design guideline of primitive nodes for prototyping multimodal AI pipelines and 2) an AI assistant, called InstructPipe, that achieves rapid prototyping of such pipelines. We gathered 236 ML pipeline proposals from 58 contributors and distilled a design space that guided our implementation space of 27 new nodes in the system. We further demonstrated InstructPipe, an assistant that allows users to prototype a pipeline through natural language instruction (Figure 1). InstructPipe presents a generated pipeline in the visual programming workspace after the user's instruction. If the generated pipeline does meet the user's requirement, the user can further work on the creativity process based on the generated pipeline by following the standard workflow of visual programming.

2 DESIGN SPACE OF PRIMITIVE NODES FOR MULTI-MODAL AI PIPELINES

To understand what pipelines people would like to build with visual programming, we conducted an online survey to gather proposals for a multi-modal AI pipeline. These results informed a new design space of primitive nodes for visual programming with language and vision models.

2.1 Online Survey

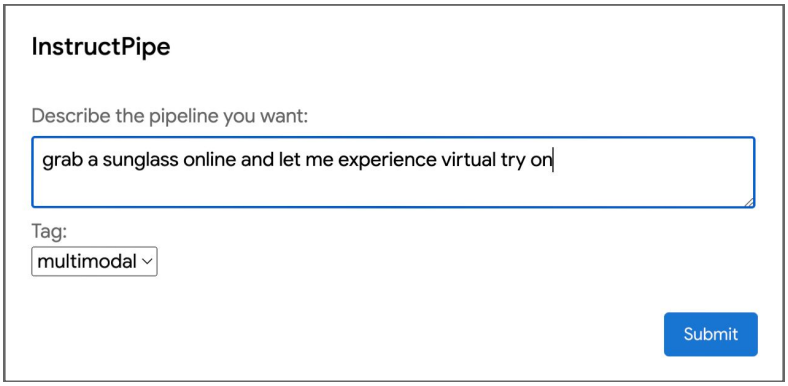
We distribute an online survey through internal communication channels, email lists, and social media. In two weeks, we collected 236 pipeline proposals from 58 respondents, of whom 32 (55%) had no prior experience with Visual Blocks. Using the affinity diagram approach, two researchers organized participants' responses and classified the proposals into three categories: language, vision, and multi-modal pipelines.

2.2 Observations

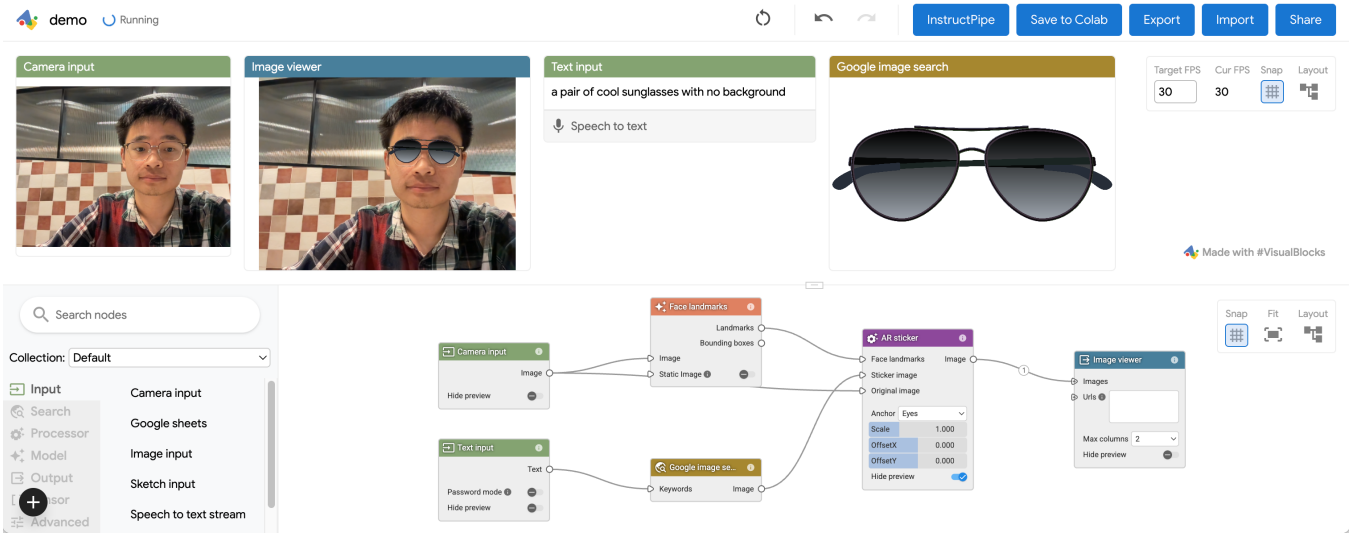
We followed the design space analysis methods [2] and held iterative discussion sessions. Through this process, we discerned that the existing 38 nodes in Visual Blocks were insufficient to satisfy the diverse requirements to meet participants' needs. We then categorized the missing nodes into the following three classes:

- **input nodes** (*e.g.*, text input, Google Sheets reader)
- **output nodes** (*e.g.*, markdown viewer, Colab output)
- **processor nodes** (*e.g.*, large language models, vision-language models)

Input and output nodes serve as protocols that extend the *creative usage* of a pipeline. Processor nodes decide the *intelligence* of data processing. For example, with a *Google Sheets reader* node, a *Large language model* node, and a *Colab* output node, one can create a



(a) The instruction dialog of InstructPipe.



(b) The visual programming interface of InstructPipe.

Figure 3: The user interface of InstructPipe. The user can first click on the "InstructPipe" button on the top-right corner of the interface in (b). A dialog will appear, and the user can input the instructions and select a category tag. InstructPipe then renders a pipeline on (b), in which the user can interactively explore and revise.

pipeline of "*Visualize the user preferences data from a Google Sheet in a bar chart*".

2.3 Node Design Matrix

Informed by the set of nodes and the classification derived from the online survey, we crafted a design space specifically focused on processor nodes, aiming to facilitate the prototyping of language and vision pipelines. As shown in Figure 2, we categorize intermediate nodes based on their input/output types including language, vision, and metadata. "Metadata", defined as the data of the data, represents intermediate features commonly used in machine learning pipelines, such as text vectors and image landmarks.

3 SYSTEM OVERVIEW

Our system upgrades Visual Blocks [6, 7] in two aspects. First, we implemented a large range of primitive nodes to support users' creativity in prototyping multimodal AI pipelines. By following the design matrix in Figure 2(a), we implemented 27 new nodes into the system to support users' prototyping experience. To further enhance users' rapid prototyping experiences, we implemented InstructPipe, an AI assistant that enables users to build pipelines by prompting it and finetuning the generated pipeline in the visual programming workspace.

3.1 Node Library

Informed by the design space, we implement new nodes that enhance the intelligence of visual programming. Figure 2(b) summarizes our node implementation. As listed in Appendix A, we integrate 27 new nodes into Visual Blocks to satisfy people's proposed pipelines. The following elaborates on five selected nodes below:

- **PaLM** [5]: a large language model (LLM) that generates text given a text prompt.
- **Google Text Search**: given a text query, output a list of URLs returned from Google.

- **Google Image Search**: given a text query, output the first image from Google.
- **PaLI** [3, 4]: a Vision-Language Model that generates text from a joint of image and text inputs.
- **Imagen** [12]: a text-to-image diffusion model.

3.2 InstructPipe — An AI Assistant for Rapid Pipeline Prototyping

InstructPipe enables users to generate a pipeline by simply providing text-based instructions [16]. We implemented InstructPipe based on Gemini Pro [9]. The raw output format of our LLM is pseudocode, which is proven to be an efficient representation of a directed graph [10, 14]. InstructPipe renders the generated directed graph on the visual programming workspace by initializing with pre-defined default property values. For example, a PaLM node is initialized with "max tokens = 256" (a property value in the PaLM node). This implies that the core task achieved by InstructPipe is selecting and connecting nodes, and it leaves the task of finetuning property value in each node to users by displaying the pipeline in the visual programming workspace.

4 USER JOURNEY

Here, we describe an exemplary user journey when using our system to prototype a multimodal AI pipeline. Different from the mainstream approach of visual programming, our demonstration provides users with a new experience of visual programming by first prompting the system and then finetuning the generated pipeline in the visual programming workspace.

4.1 Step I: Describing a Target Pipeline

The user can first click on the "InstructPipe" button on the top-right corner of the interface (Figure 3b). The system then triggers a dialog (Figure 3a) for the user to write a prompt and select a tag. The tag can be "language", "visual", or "multimodal". After the user clicks "Submit" at the bottom-right corner of the dialog, it completes the "instruction" process to the AI assistant.

4.2 Step II: Finetuning Generated Results in the Visual Programming Workspace

After the instruction, InstructPipe generates a pipeline in the visual programming workspace. If the user is not satisfied with the generation, they can further work on the pipeline by following the standard approach of visual programming, *i.e.*, selecting and connecting nodes, based on the generated results. The visual programming workspace also provides the user with a platform to explore the optimal property value in each node for the most satisfactory results.

5 CONCLUSION

In this paper, we introduced a visual programming system powered by a wide range of multi-model nodes and InstructPipe, an AI assistant that facilitates users to build pipelines with instructions. The node implementation in the system was informed by our design guideline, which is based on 236 pipeline proposals from 58 participants. InstructPipe provides users new experience in prototyping visual programming pipelines by firstly writing an instruction and then perform finetuning in the visual programming workspace. We envision that InstructPipe aspires to streamline workflow processes for novices in ML pipeline development and inspire novel applications through its user-friendly interface.

REFERENCES

[1] 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Stuart K Card, Jock D Mackinlay, and George G Robertson. 1991. A Morphological Analysis of the Design Space of Input Devices. *ACM Transactions on Information Systems (TOIS)* 9, 2 (1991), 99–122. https://doi.org/10.1145/123078.128726

[3] Xi Chen, Josip Djolonga, Piotr Padlewski, Basil Mustafa, Soravit Changpinyo, Jialin Wu, Carlos Riquelme Ruiz, Sebastian Goodman, Xiao Wang, Yi Tay, Siamak Shakeri, Mostafa Dehghani, Daniel Salz, Mario Lucic, Michael Tschannen, Arsha Nagrani, Hexiang Hu, Mandar Joshi, Bo Pang, Ceslee Montgomery, Paulina Pietrzyk, Marvin Ritter, AJ Piergiovanni, Matthias Minderer, Filip Pavetic, Austin Waters, Gang Li, Ibrahim Alabdulmohsin, Lucas Beyer, Julien Amelot, Kenton Lee, Andreas Peter Steiner, Yang Li, Daniel Keysers, Anurag Arnab, Yuanzhong Xu, Keran Rong, Alexander Kolesnikov, Mojtaba Seyedhosseini, Anelia Angelova, Xiaohua Zhai, Neil Houlsby, and Radu Soricut. 2023. PaLI-X: On Scaling Up a Multilingual Vision and Language Model. arXiv:2305.18565 [cs.CV]

[4] Xi Chen, Xiao Wang, Soravit Changpinyo, AJ Piergiovanni, Piotr Padlewski, Daniel Salz, Sebastian Goodman, Adam Grycner, Basil Mustafa, Lucas Beyer, Alexander Kolesnikov, Joan Puigcerver, Nan Ding, Keran Rong, Hassan Akbari, Gaurav Mishra, Linting Xue, Ashish Thapliyal, James Bradbury, Weicheng Kuo, Mojtaba Seyedhosseini, Chao Jia, Burcu Karagol Ayan, Carlos Riquelme, Andreas Steiner, Anelia Angelova, Xiaohua Zhai, Neil Houlsby, and Radu Soricut. 2023. PaLI: A Jointly-Scaled Multilingual Language-Image Model. arXiv:2209.06794 [cs.CV]

[5] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2023. PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research* 24, 240 (2023), 1–113. http://jmlr.org/papers/v24/22-1144.html

[6] Ruofei Du, Na Li, Jing Jin, Michelle Carney, Scott Miles, Maria Kleiner, Xiuxiu Yuan, Yinda Zhang, Anuva Kulkarni, Xingyu Liu, Ahmed Sabie, Sergio Orts-Escolano, Abhishek Kar, Ping Yu, Ram Iyengar, Adarsh Kowdle, and Alex Olwal. 2023. Rapsai: Accelerating Machine Learning Prototyping of Multimedia Applications Through Visual Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 125, 23 pages. https://doi.org/10.1145/3544548.3581338

[7] Ruofei Du, Na Li, Jing Jin, Michelle Carney, Xiuxiu Yuan, Kristen Wright, Mark Sherwood, Jason Mayes, Lin Chen, Jun Jiang, Jingtao Zhou, Zhongyi Zhou, Ping Yu, Adarsh Kowdle, Ram Iyengar, and Alex Olwal. 2023. Experiencing Visual Blocks for ML: Visual Prototyping of AI Pipelines. In *Adjunct Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (San Francisco, CA, USA) *(UIST '23 Adjunct)*. Association for Computing Machinery, New York, NY, USA, Article 76, 3 pages. https://doi.org/10.1145/3586182.3615817

[8] GitHub. 2023. GitHub Copilot · Your AI Pair Programmer. https://github.com/features/copilot

[9] Gemini Team Google. 2023. Gemini: A Family of Highly Capable Multimodal Models. https://doi.org/10.48550/arXiv.2312.11805 arXiv:2312.11805 [cs.CL]

[10] Tanmay Gupta and Aniruddha Kembhavi. 2023. Visual Programming: Compositional Visual Reasoning Without Training. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. arXiv. https://doi.org/10.48550/arXiv.2211.11559

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems* 32 (2019). https://doi.org/10.48550/arXiv.1912.01703

[12] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. 2022. Photorealistic Text-to-Image Diffusion Models With Deep Language Understanding. arXiv:2205.11487 [cs.CV]
[13] Ray Smith, Daria Antonova, and Dar-Shyang Lee. 2009. Adapting the Tesseract open source OCR engine for multilingual OCR. In *Proceedings of the International Workshop on Multilingual OCR*. 1–8.
[14] Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. ViperGPT: Visual Inference via Python Execution for Reasoning. arXiv:2303.08128 [cs.CV]
[15] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's Transformers: State-of-the-Art Natural Language Processing. *ArXiv Preprint ArXiv:1910.03771* (2019). https://arxiv.org/pdf/1910.03771
[16] Zhongyi Zhou, Jing Jin, Vrushank Phadnis, Xiuxiu Yuan, Jun Jiang, Xun Qian, Jingtao Zhou, Yiyi Huang, Zheng Xu, Yinda Zhang, Kristen Wright, Jason Mayes, Mark Sherwood, Johnny Lee, Alex Olwal, David Kim, Ram Iyengar, Na Li, and Ruofei Du. 2023. InstructPipe: Building Visual Programming Pipelines with Human Instructions. https://doi.org/10.48550/arXiv.2312.09672 arXiv:2312.09672 [cs.HC]

APPENDIX

A A LIBRARY OF NEWLY IMPLEMENTED PRIMITIVE NODES

Drawing upon §2 Design Space, we implement the full list of 27 new nodes that have been integrated into the most recent iteration of Visual Blocks. These additions were formulated based on insights garnered from the community survey, as detailed below:

(1) **PaLM** [5]: a large language model (LLM).
(2) **Google Text Search**: given a text query, output a list of URLs returned from Google.
(3) **Google Image Search**: given a text query, output the first image from Google.
(4) **PaLI** [3, 4] (a Search node): Pathways Language and Image model that generates text from a joint of image and text inputs.
(5) **Imagen** [12]: a text-to-image diffusion model.
(6) **OCR** [13]: a text recognition model from an image query.
(7) **Colab Output***: given input code, output to a Colab node to execute the code.
(8) **URL to HTML**: given an URL as text, fetch the webpage as text output.
(9) **URL to Image***: given an URL as text, fetch the data as an image output.
(10) **Markdown Viewer**: given a text in MarkDown language, visualize it as a webpage.
(11) **Text Processor**: given multiple text, join them together or format them in a certain way.
(12) **String Picker**: given a list of text and an ID, fetch one of them as output.
(13) **HTML Viewer**: given a text in HTML language, visualize it as a webpage.
(14) **Hand Pose Detection**: given an image, return landmarks of the hands using MediaPipe.
(15) **Hand Gesture Detection**: given an image, recognize the gesture using MediaPipe.
(16) **Face Landmark Detection**: given an image, return landmarks of the face.
(17) **Body Segmentation**: run a deployed MediaPipe body segmentation model.
(18) **Object Detection***: given an image, output the detected object class name and their bounding boxes.
(19) **Bounding Box Visualizer***: visualizing bounding boxes of machine learning models.
(20) **Landmark Visualizer**: visualizing pose landmarks of machine learning models.
(21) **Mask Visualizer**: visualizing segmentation mask of machine learning models.
(22) **Virtual Sticker**: given landmarks, an original image, and a sticker image, overlay sticker image onto the original image upon the given landmark position.
(23) **Text Toxicity***: given a text input, output the toxicity level of the language.
(24) **Text Processor**: given (a) text input(s), output a reformated text.
(25) **Google Sheets Reader**: given a Google Sheets URL and range of data, fetch data and output as text.
(26) **Google Sheets Writer***: given a Google Sheets URL and range of data, output the input text directly to the Google Sheets.
(27) **Custom API***: Given an API URL, query it via GET or POST methods and fetch the result.

Nodes marked with * are out of the scope of the current version of InstructPipe.