TEXT2LEARN

PROJECT REPORT

Srivats Poddar
New York University - Courant
sp7811@nyu.edu

Rahul Raman
New York University - Courant
rr4549@nyu.edu

Pranav Thorat
New York University - Courant
pat9991@nyu.edu

May 15, 2024

ABSTRACT

This paper introduces Text2Learn, a pioneering framework designed to create educational videos using Manim, a Python library for mathematical animation (The Manim Community Developers [2024]). Addressing the challenge of translating user prompts into actionable code, Text2Learn leverages the coding capabilities of Large Language Models (LLMs) to streamline the process. While LLMs exhibit remarkable creativity across various domains, they often struggle to generate concise and purposeful code, especially for educational content. Our approach enhances LLMs with Chain of Thoughts (CoT) (Wei et al. [2023]), establishing a hierarchical plan to (i) split videos into scenes, (ii) ideate objects required for visualizing each scene, (iii) lay out detailed steps defining animations for these objects, and (iv) generate Manim code for each scene. Our experiments demonstrate that our proposed framework can generate scene-by-scene video plans relevant to the topic, ensuring coherent progression and adherence to the generated plan. Additionally, it successfully produces error-free, relevant, executable code, delivering a complete video without any human intervention. We also propose that we can further our framework to dynamically control the strength of positioning guidance and generate animations with image references. We hope our framework inspires future work on better integrating the planning abilities of LLMs into consistent long videos structured with multiple animation scenes. The code for our project is available on GitHub.

Keywords Large Language Models · Chain of thoughts · text-to-video · manim

1 Introduction

Large Language Models (LLMs) such as GPT4(OpenAI et al. [2024]) and Llama 3 (Meta [2024]) have showcased impressive capabilities in generating code tailored to user prompts. Despite their proficiency in conceptual understanding and task planning, their application is limited by context size, particularly in generating large and well-structured code (Wei et al. [2023]). In the realm of Manim, where animations are constructed frame by frame, the complexity arises from the necessity to coordinate multiple frames to depict a coherent scene (Dettmers et al. [2022]). Educational video production demands meticulous planning and organization, a challenge for LLMs which often prioritize object quality over scene coherence (Besta et al. [2024]). Building upon prior methodologies like Chain of Thought (CoT) and Graph of Thoughts, our work seeks to enhance LLMs' problem-solving capabilities by structuring the reasoning process into a coherent graph, thereby improving prompt understanding and response quality (Wei et al. [2023], Long [2023], Yao et al. [2023], Xie et al. [2023]). Additionally, drawing from the Planning Definition and Domain Language (PDDL) paradigm, we emphasize maintaining structured prompt-response formats to enhance explainability and utilize the Retrieve and Generate (RAG) approach for concept clarification through example retrieval (Guan et al. [2023], Lewis et al. [2021]).

Furthermore, LLMs have demonstrated proficiency in generating layouts and programs, particularly in controlling downstream visual modules for scene description and object positioning (Cho et al. [2023], Lin et al. [2023], Zala et al. [2023]). Our research focuses on harnessing these capabilities to facilitate the generation of Manim code for educational videos. By delineating a hierarchical plan encompassing scene description, object placement, animation sequencing, and code generation, our proposed framework, Text2Learn, aims to bridge the gap between LLM creativity

and structured code output. In contrast to existing approaches like Chain of Thoughts, Text2Learn provides a systematic methodology tailored specifically for the intricate task of video production from textual prompts, thereby contributing to the advancement of educational video generation.

In this paper, we present empirical evaluations demonstrating the efficacy of our framework in generating Manim videos. Through comparisons with standard prompting methods, we showcase the superiority of Text2Learn in terms of code quality, scene coherence, and overall video production efficiency. By leveraging the inherent capabilities of LLMs in conjunction with structured planning methodologies, our framework offers a promising avenue for enhancing the automation and efficiency of educational video production, thus addressing a significant research gap in the field.

Our main contributions are: (i) successfully generating a detailed and coherent 'video plan' that includes comprehensive steps for executing animations, object positions, and entity/background consistency groupings to guide downstream video generation (Section 3); (ii) demonstrating that, given this detailed plan, our framework can convert it into error-free, relevant, and executable python code; and (iii) providing qualitative examples, with an appendix detailing our prompt choices (Section 6).

2 Related Work

Multi-scene video generation via LLM-guided planning (Lin et al. [2023]) given a single text prompt, query video planner LLM (GPT-4) to expand it into a 'video plan'. Next, Layout-guided text-to-video generation (Layout2Vid) generates a video. As described in Self-planning Code Generation with Large Language Models (Jiang et al. [2023]), improved code generation using LLMs by introducing planning into code generation to help the model understand complex intent and reduce the difficulty of problem-solving.The plan obtained and the intent is used to guide LLM in generating code. Latent Predictor Networks for Code Generation [14], paper uses Traditional code generation approaches which are based on supervised learning, which initially treats code as equivalent to natural language and trains a neural network named Latent Prediction Network.

In (Wan et al. [2024]), they have explored the realm of knowledge fusion for LLMs to create a unified model that combines the capabilities of multiple LLMs, introducing FUSELLM, which leverages the generative distributions of these source LLMs to externalize their knowledge and employs them in the continual training the target LLM. The text-to-video (T2V) generation task is to generate videos from text descriptions. Early T2V generation models ((Li et al. [2017, 2019])) used variational autoencoders (VAE) (Kingma and Welling [2022]) and generative adversarial networks (GAN) (Goodfellow et al. [2014]), while multimodal language models (Hong et al. [2022], Wu et al. [2022], Maharana et al. [2022], Ge et al. [2022], Ho et al. [2022a]) and denoising diffusion models (Ho et al. [2022a], Blattmann et al. [2023], Khachatryan et al. [2023], Yin et al. [2023]) have become popular for recent works. Since training a T2V generation model from scratch is computationally expensive, recent work often lever ages pre-trained text-to-image (T2I) generation models such as Stable Diffusion (Rombach et al. [2022]) by finetuning them on text-video pairs (Wang et al. [2023], Blattmann et al. [2023]). While this warm-start strategy enables high-resolution video generation, it comes with the limitation of only being able to generate short video clips, as T2I models lack the ability to maintain consistency through long videos. Recent works on long video generation (Villegas et al. [2022]) aim at generating long videos of a few minutes. However, the generated videos often display the continuation or repetitive patterns of a single action (e.g., driving a car) instead of transitions and dynamics of multiple changing actions/events (e.g., five steps about how to bake a cake).

Early works (Pan et al. [2018]) are mainly focused on video generation in simple domains, such as moving digits or specific human actions. To our knowledge, Sync-DRAW [T] is the first T2V generation approach that leverages a VAE with recurrent attention. (Pan et al. [2018]) and (Li et al. [2018]) extend GANs from image generation to T2V generation. More recently, GODIVA (Wu et al. [2021a]) is the first to use 2D VQVAE and sparse attention for T2V generation supporting more realistic scenes. N ¨ UWA (Wu et al. [2021b]) extends GODIVA, and presents a unified representation for various generation tasks in a multitask learning scheme. To further improve the performance of T2V generation, CogVideo (Hong et al. [2022]) is built on top of a frozen CogView-2 T2I model by adding additional temporal attention modules. Video Diffusion Models (VDM) (Ho et al. [2022b]) uses a space-time factorized U-Net with joint image and video data training. While both CogVideo and VDM collected 10M private text-video pairs for training, our work uses solely open-source datasets, making it easier to reproduce.

3 Methodology

As illustrated in Figure 1, the LLM is the planner and provides a detailed animation plan from a single text prompt for downstream Manim code generation. The animation plan consists of four steps: (1) Video Plan by Scenes, (2) Scene-wise Object Planning, (3) Scene-wise Detailed Steps, and (4) Scene-wise Code Generation.

3.1 Video Plan by Scenes

This is the first step in our pipeline. With a single-line text prompt, we use the LLM to generate the scene planning required to encapsulate the idea of the text prompt. For this step, we first generate a brief description of the overall animation and then break it into short individual scenes. This provides the preliminary animation guidelines, adhered to in the subsequent generation stage for detailed animation steps.

3.2 Scene-wise Object Planning

In the second step, given the general animation context of a scene, a list of objects is generated that are required for this animation scene. For each object, we generate the `label`, `description`, and `position` of the object in the scene. Each object position is in the format `[x, y, height, width]`, inspired by DiagrammerGPT (Zala et al. [2023]), to meet the requirements of the Manim coordinate system.

3.3 Scene-wise Detailed Steps

Provided with the general animation description for the scene along with the objects required, in this step, the LLM generates the detailed step-by-step instructions required to accurately animate the given scene. The LLMs have demonstrated exceptional conceptual knowledge in the domain of computer science, thereby being able to construct intuitive examples and provide immaculate steps for animation.

3.4 Scene-wise Code Generation

With the detailed animation steps, the Manim code is generated in the final step of animation layout generation. By providing the LLM with a comprehensive scene plan and precise instructions, we aim to focus on generating simplistic animations and problem-solving abilities for creating accurate and efficient Python code to animate educational videos rather than getting carried away with creativity.

3.5 Error Code Correction Loop

Recognizing that LLM-generated code may contain errors in the initial attempts, we employ an iterative approach to refine the code quality. This error correction loop entails sending back the erroneous code along with the error encountered to the LLM to rectify the issue. Specifically, we provide the LLM with the error stack trace to pinpoint the location and nature of the error, enabling more targeted corrections. This iterative process is repeated for a predefined number of iterations, typically capped at 5 attempts, to balance computational and cost efficiency with code refinement. By leveraging this error correction loop, we aim to iteratively enhance the accuracy and correctness of the generated Manim code, ensuring the production of high-quality educational videos.

4 Experiment Setup

We evaluate our framework on multiple LLMs such as Llama 3, Instruct-Llama 3, GPT3.5-Turbo and GPT4. Llama 3 and Instruct-Llama 3 are 4-bit quantized models and computed in half-precision (`torch.float16`) using bitsandbytes(Dettmers et al. [2022]) and deployed on NYU HPC V100 GPU (16 GB VRAM). Through extensive empirical and human evaluations, we conclude that GPT4 better adhered to our pipeline setup. We illustrate our text-to-code rendered video generation in Figure 2. Our method generates consistent animation scenes and explainable videos with scenes up to a maximum of 1 minute in length.

We have also experimented on a small scale to integrate RAG with our current pipeline. Currently, this is **not integrated** with the current pipeline. For our use-case, we hypothesize to utilize RAG in the following way. Firstly, we aim to store images and their corresponding captions in the database. To store and index our splits, so that they can later be searched over, this is often done using a VectorStore and Embeddings model. This will support during object planning to improve the quality of animation frames. Secondly, we also aimed to use the RAG database with relevant technical corpus to get context-accurate responses. To provide a proof-of-concept on the working of RAG, we instantiate a 'RetrievalQA' chain over a vectorstore retriever, from the 'LangChain' package which takes as parameters: a retriever, an LLM model, and a chain type as input arguments. The model which we used is: "TheBloke/Llama-2-13b-Chat-GPTQ" (which is Meta's Llama2 made available via Hugging Face). RetrievalQA chain actually uses "load-qa-chain" under the hood. When the QA chain receives a query, the retriever retrieves information related to the query from the vector store (such as langchain's "vectorstores" Chroma or SAISS which is one of the most common ways to store and search over unstructured data to embed it using the "HuggingFaceEmbeddings" and store the resulting embedding vectors, and

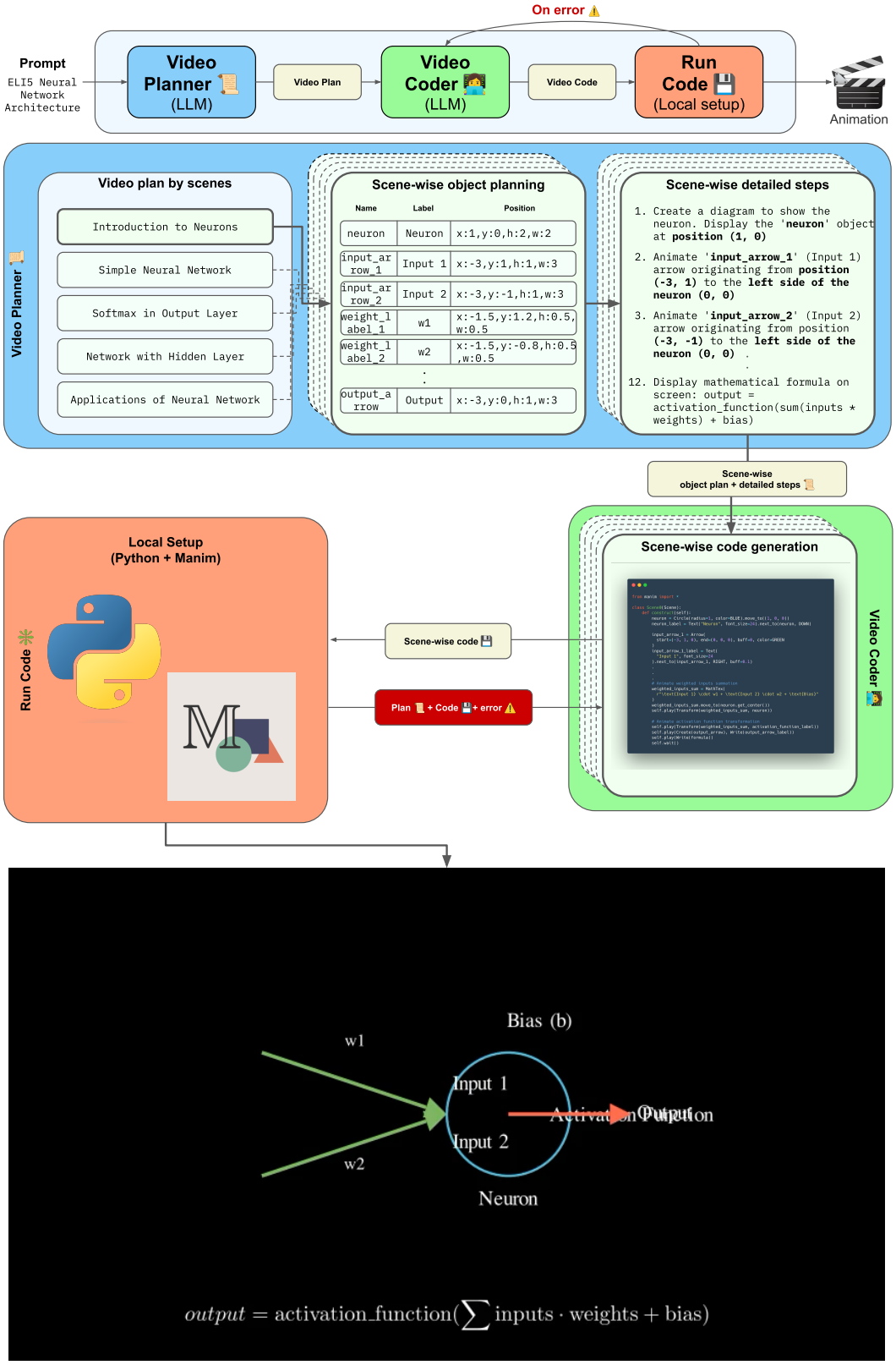


Figure 1: Text2Learn pipeline

then at query time to embed the unstructured query and retrieve the embedding vectors that are 'most similar' to the embedded query). The LLM then generates the text or response from the retrieved documents. For creating a prompt, we use the Llama 2 prompt structure and LangChain's "PromptTemplate". Efficient use of the LLM prompt engineering, allows us to pass in the recommended prompt structure for Llama 2 for the QA.

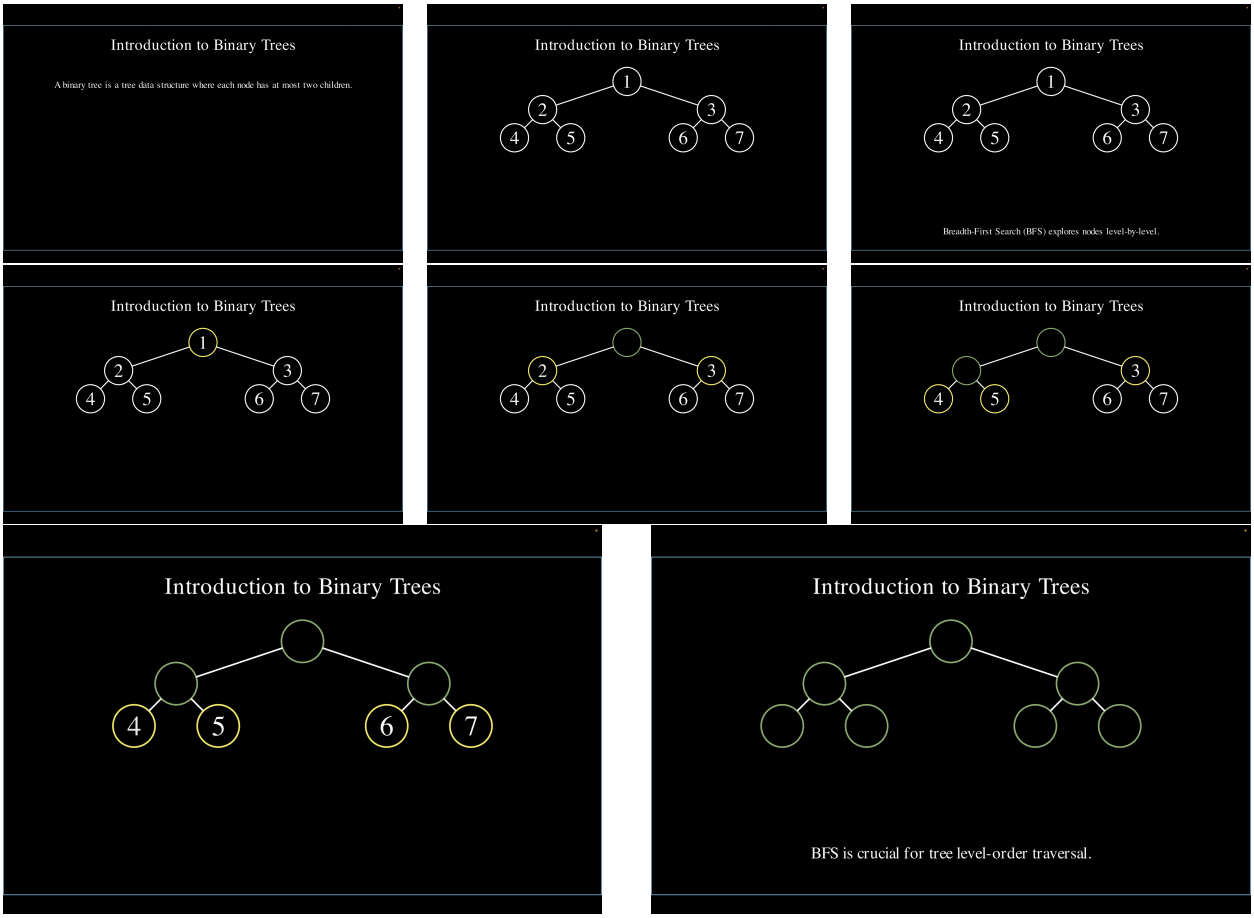


Figure 2: A series of frames from a scene generated for the prompt: "*Explain BFS traversal in a binary tree*". From left-to-right, top-to-bottom, the following steps can be observed: (i) GPT4 introduces the topic by giving a title and definition for binary trees. (ii) It visualizes a binary tree and (iii) defines BFS traversal. (iv) The root node is highlighted in yellow to depict the 'enqueuing' operation in BFS. (v) The first node in the queue is 'dequeued', with its children enqueued, and marked as 'visited' by highlighting in green and fading out the node value. (vi-viii) These steps are repeated until all nodes are visited.

5 Conclusion

In this work, we propose Text2Learn, a novel chain-of-thoughts framework for educational video generation that leverages the capabilities of LLMs for scene-wise animation planning and corresponding Manim code generation. The hierarchical pipeline begins by breaking the user prompt into scenes and providing brief descriptions of each scene's content. This is followed by identifying the required objects for each scene, generating animation guidelines for each scene, and finally producing Manim code to render videos based on these guidelines and objects. Our experiments demonstrate that the Text2Learn framework successfully produces error-free, executable code, delivering a complete video with zero human intervention. We hope our work inspires further exploration into integrating the planning capabilities of LLMs into complete topic-explanation video generation.

References

The Manim Community Developers. Manim – Mathematical Animation Framework, April 2024. URL `https:`

//www.manim.community/.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024.

Meta. Llama3, April 2024. URL https://ai.meta.com/blog/meta-llama-3/.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16): 17682–17690, March 2024. ISSN 2159-5399. doi:10.1609/aaai.v38i16.29720. URL http://dx.doi.org/10.1609/aaai.v38i16.29720.

Jieyi Long. Large language model guided tree-of-thought, 2023.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, Xu Zhao, Min-Yen Kan, Junxian He, and Qizhe Xie. Self-evaluation guided beam search for reasoning, 2023.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning, 2023.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

Jaemin Cho, Abhay Zala, and Mohit Bansal. Visual programming for text-to-image generation and evaluation, 2023.

Han Lin, Abhay Zala, Jaemin Cho, and Mohit Bansal. Videodirectorgpt: Consistent multi-scene video generation via llm-guided planning, 2023.

Abhay Zala, Han Lin, Jaemin Cho, and Mohit Bansal. Diagrammergpt: Generating open-domain, open-platform diagrams via llm planning, 2023.

Xue Jiang, Yihong Dong, Lecheng Wang, Zheng Fang, Qiwei Shang, Ge Li, Zhi Jin, and Wenpin Jiao. Self-planning code generation with large language models, 2023.

Fanqi Wan, Xinting Huang, Deng Cai, Xiaojun Quan, Wei Bi, and Shuming Shi. Knowledge fusion of large language models, 2024.

Yitong Li, Martin Renqiang Min, Dinghan Shen, David Carlson, and Lawrence Carin. Video generation from text, 2017.

Yitong Li, Zhe Gan, Yelong Shen, Jingjing Liu, Yu Cheng, Yuexin Wu, Lawrence Carin, David Carlson, and Jianfeng Gao. Storygan: A sequential conditional gan for story visualization, 2019.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

Wenyi Hong, Ming Ding, Wendi Zheng, Xinghan Liu, and Jie Tang. Cogvideo: Large-scale pretraining for text-to-video generation via transformers, 2022.

Chenfei Wu, Jian Liang, Xiaowei Hu, Zhe Gan, Jianfeng Wang, Lijuan Wang, Zicheng Liu, Yuejian Fang, and Nan Duan. Nuwa-infinity: Autoregressive over autoregressive generation for infinite visual synthesis, 2022.

Adyasha Maharana, Darryl Hannan, and Mohit Bansal. Storydall-e: Adapting pretrained text-to-image transformers for story continuation, 2022.

Songwei Ge, Thomas Hayes, Harry Yang, Xi Yin, Guan Pang, David Jacobs, Jia-Bin Huang, and Devi Parikh. Long video generation with time-agnostic vqgan and time-sensitive transformer, 2022.

Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P. Kingma, Ben Poole, Mohammad Norouzi, David J. Fleet, and Tim Salimans. Imagen video: High definition video generation with diffusion models, 2022a.

Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your latents: High-resolution video synthesis with latent diffusion models, 2023.

Levon Khachatryan, Andranik Movsisyan, Vahram Tadevosyan, Roberto Henschel, Zhangyang Wang, Shant Navasardyan, and Humphrey Shi. Text2video-zero: Text-to-image diffusion models are zero-shot video generators, 2023.

Shengming Yin, Chenfei Wu, Huan Yang, Jianfeng Wang, Xiaodong Wang, Minheng Ni, Zhengyuan Yang, Linjie Li, Shuguang Liu, Fan Yang, Jianlong Fu, Gong Ming, Lijuan Wang, Zicheng Liu, Houqiang Li, and Nan Duan. Nuwa-xl: Diffusion over diffusion for extremely long video generation, 2023.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

Jiuniu Wang, Hangjie Yuan, Dayou Chen, Yingya Zhang, Xiang Wang, and Shiwei Zhang. Modelscope text-to-video technical report, 2023.

Ruben Villegas, Mohammad Babaeizadeh, Pieter-Jan Kindermans, Hernan Moraldo, Han Zhang, Mohammad Taghi Saffar, Santiago Castro, Julius Kunze, and Dumitru Erhan. Phenaki: Variable length video generation from open domain textual description, 2022.

Yingwei Pan, Zhaofan Qiu, Ting Yao, Houqiang Li, and Tao Mei. To create what you tell: Generating videos from captions, 2018.

Yitong Li, Martin Min, Dinghan Shen, David Carlson, and Lawrence Carin. Video generation from text. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. doi:10.1609/aaai.v32i1.12233. URL `https://ojs.aaai.org/index.php/AAAI/article/view/12233`.

Chenfei Wu, Lun Huang, Qianxi Zhang, Binyang Li, Lei Ji, Fan Yang, Guillermo Sapiro, and Nan Duan. Godiva: Generating open-domain videos from natural descriptions, 2021a.

Chenfei Wu, Jian Liang, Lei Ji, Fan Yang, Yuejian Fang, Daxin Jiang, and Nan Duan. Nüwa: Visual synthesis pre-training for neural visual world creation, 2021b.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video diffusion models, 2022b.

6 Appendix

This section mostly showcases the specific details of our prompts and our future experimental enhancements. All results discussed are empirical and based on human evaluations.

6.1 Enforcing object coordinates

Explicitly enforcing the LLM to determine the object coordinates substantially improves the object positioning in the scene.

OBJECT_PLAN_PROMPT:

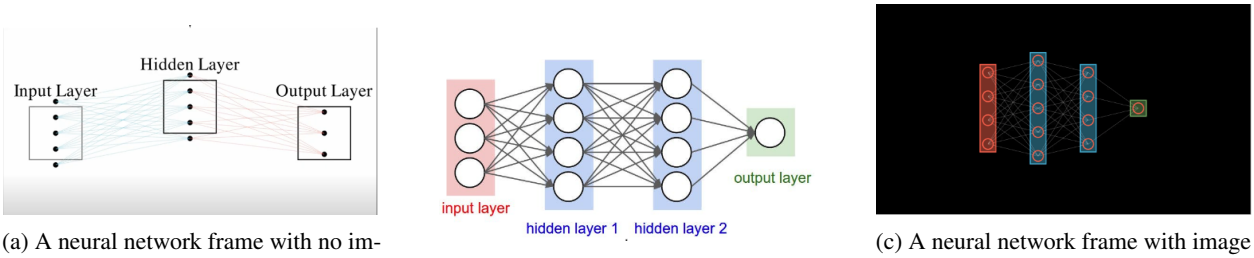
```
.
.
Return the response in below mentioned JSON format. You will be talking to an API,
     that expects just this JSON, not a human:
{
    "id": "<scene_id>",
    "instructions": "<brief description of the scene>",
    "objects": [
        {
            "id": "<object_id>",
            "name": "<object name>",
            "label": "<object label to be displayed on screen>",
            "description": "<object description>",
            "position": {
                "x": <float, x coordinate>,
                "y": <float, y coordinate>,
                "height": <float, object height>,
                "width": <float, object width>
            }
        }
        .
        .
        <List of objects>
    ]
}

Instructions to be followed
  .
  .
- -7 <= x <= 7 and -4 <= y <= 4
- The x coordinate + the width of an entity should not exceed 7.
- The y coordinate + the height of an entity should not exceed 4.
- Objects should not overlap.
  .
  .
```

6.2 Reference image during object planning

Providing a reference image improves the quality of the animation frame generated, but this could also affect the animation consistency based on the complexity of the reference image provided. This opens up an exploration possibility to retrieve the images can be from a RAG database comprising of image captions, and later retrieve the top K image captions along with their image.

6.3 Represent scene as object occupancy matrix

- Inserting the matrix as the input. In Manim, the coordinates vary from (-7, 7) on the x-axis and (-4, 4) on the y-axis. For granular measurements, the x and y axis are further divided into 10 cells each making the range from (-40, 40) and (-70, 70). The matrix in general will mostly be sparse, the major drawback is that the number of tokens required to represent this matrix is around 11,000 which significantly increases the cost of the prompt while also possibly losing the contextual meaning of the data over such long sequences.



(a) A neural network frame with no image reference

(b) reference image provided to LLM

(c) A neural network frame with image reference

Figure 3: Improvement of animation scene using reference images

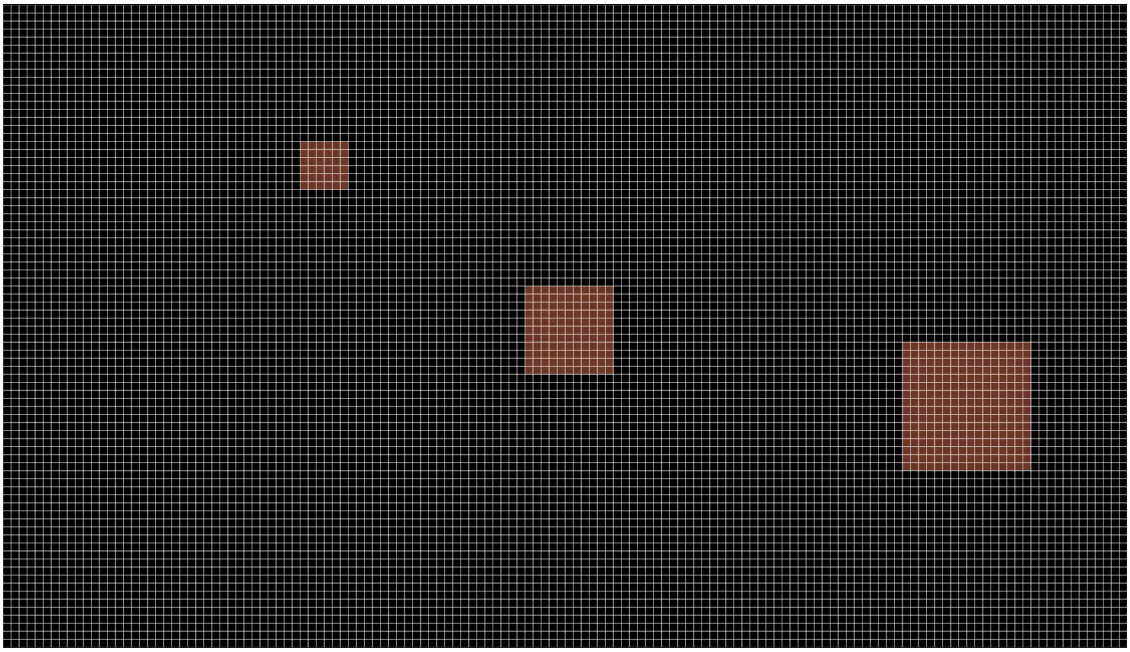


Figure 4: A screenshot of a 1920×1080 video divided into 140×80 cells. Red regions would signify occupancy.

- Possible research direction - Provide It with helper functions specifically curated for the object positioning and equip LLMs with a component that retrieves coordinates and occupancy matrices for the set of objects to enhance their capability of solving the relative positioning of objects in the scene.

6.4 Enhance error correction loop

LLM specific - while GPT4 is able to fix the errors in the code provided the errors encountered during executing the code, for Llama3 additional information such as the function signature of the error function is required. We can further provide supporting information such as the function docstring to further improve the capability of LLM to fix the errors.

For instance, if an error encountered when executing the python code and it is pointing to an issue is due to misuse of AddTextLetterByLetter function. Then provide the function details in the error correction loop (along with the Python code + STACK_TRACE_ERROR).

```
AddTextLetterByLetter : (text: 'Text', suspend_mobject_updating: 'bool' = False,
    int_func: 'Callable[[np.ndarray], np.ndarray]' = <ufunc 'ceil'>, rate_func: '
    Callable[[float], float]' = <function linear at 0x7b8f77d44a60>, time_per_char:
     'float' = 0.1, run_time: 'float | None' = None, reverse_rate_function=False,
    introducer=True, **kwargs) -> 'None'
Usage: Show a :class:`~.Text` letter by letter on the scene.

    Parameters
    ----------
    time_per_char
```

```
        Frequency of appearance of the letters.

    .. tip::

        This is currently only possible for class:`~.Text` and not for class:`~.
    MathTex`
```

6.5 Feedback loop to improve object positioning

With GPT-4's capability to take images as input, one of our experiment is to fix the overlapping object positioning issue by providing the frame of the animation scene and along with the ground-truth details of the object positioning and the instructions required to generate this animation. Through extensive testing, we observed that GPT-4 is not consistent in resolving issues through this process, and no significant progress was made.