

CREATION OF A TOUCH DESIGNER TOOL FOR VISUALS USING LARGE LANGUAGE MODELS

SANDRA GOMEZ VILLAREAL, HÈCTOR TRUJILLO RUIZ

Mater of Digital arts and creative technologies
Digital arts and creative technology
La sede
Universitat Ramon Llull

2025-2026

SUMMARY

SUMARIO

ACKNOWLEDGMENTS

0.1 HÈCTOR

First of all, I want to express my sincere gratitude to my advisor for this paper, Daniel Sabio. Thank you for the trust you placed in us and de dedication you put in helping with all the problems. Having such an artist and following example guiding us it has been an enormous pleasure.

Gerard, Jesús, thank you for your unconditional support with the project. You are both fundamental parts of my professional and personal life.

Sandy, nakama, compartir este proyecto contigo ha sido un reto tanto estresante como increíble. Nunca hubiera imaginado hacer un trabajo así con alguien mas y ha sido un auténtico placer. Te has convertido en una más del equipo y espero que tanto estando aquí o en México, sigamos manteniendo esta bonita amistad.

Mariona, sabíem que aquest repte seria complicat, però l'hem aconseguit. Gràcies per ser-hi sempre, per ser el meu pilar i per encoratjar-me a seguir endavant.

Finalment, no em puc oblidar de les persones que han fet possible tot això: Eva i Carles. Sense vosaltres, realitzar aquest treball hauria estat impossible. Gràcies per l'educació que m'heu donat, per confiar en mi, per facilitar-me tots els recursos per poder estudiar el que volia i pel vostre suport incondicional. Moltes gràcies de tot cor.

CONTENTS

0.1 Hèctor	iv
1 INTRODUCTION	1
1.1 Motivation and research problem	1
1.2 Hypothesis and objectives	2
2 THEORETICAL FUNDATIONS	4
2.1 Visual history	4
2.2 Touch Designer	7
2.3 LOPS	8
2.4 Large Language models (LLM)	11
3 RESEARCH	13
3.1 Introduction	13
3.2 Training of the LLM	14
3.2.1 First approach	15
3.2.2 LOPS	17
3.3 Tool Creation	24
3.3.1 Orquestration	25
3.3.2 Database Import & semantic query	27
3.3.3 Prompt code Generation	30
3.3.4 OpenGL Shading Language (GLSL) shader	31
4 PROJECT PLANNING AND ECONOMIC STUDY	36
4.1 Project planning and management	36
4.2 Economic study of the work perfomed	38
5 CONCLUSIONS	40
5.1 Conclusions	40
A TIPS AND GUIDES FOR THE SETUP	41
A.1 How to connect an agent with memory	41
A.2 How to introduce your information docs to the agent	43
A.3 Problem with Retrieval Augmented Generation (RAG) Index chunks	44
B CODES USED FOR THE PROJECT	49
B.1 Papers_clean.py	49
B.2 PapersEmbedding.py	49
BIBLIOGRAPHY	52

LIST OF FIGURES

Figure 1	Similarities between visuals in concerts and Touch Designer tutorials, focused on simple geometrical shapes with audioreactivity	2
Figure 2	Sony Portapak camera (Maison de la Vidéo & du Cinéma: [11])	5
Figure 3	Images of Gaudi's Architecture	14
Figure 4	Example of a Comma Separated Values (CSV) with the structure of the project	16
Figure 5	Place to select the Folder with all the documents to be embedded by the Large Lenguaje Model (LLM)	18
Figure 6	Options Parse All Documents, Parse Single File and Clear Tables	19
Figure 7	Connection between Source Docs and RAG index	20
Figure 8	Options of the RAG index	21
Figure 9	Index created	22
Figure 10	Connecting the RAG index into the RAG retriever.	23
Figure 11	Diagram of how a Multi-Query Retrieval works	23
Figure 12	Adding the RAG retriever into the agent.	24
Figure 13	Diagram of the system's workflow.	25
Figure 14	How to add Agents to the Agent Orquestrator.	26
Figure 15	How to add the validator agent and the Task Data Operator (DAT).	27
Figure 16	Extra options of the agent orquestrator.	27
Figure 17	Interface to write how the agent must behave.	29
Figure 18	Image of te workflow between two source agents and the Orquestrators.	30
Figure 19	Prompt made by the Prompt Agent in order to create the GLSL shader.	31
Figure 20	Orders given to the Agent_shader.	32
Figure 21	Optional chat added in case the primary workflow do not work.	33

Figure 22	Model Context Protocol (MCP) server added to the agent to be able to manipulate Touch Designer components.	34
Figure 23	Shaders generated automatically.	34
Figure 24	Shaders generated Manually after the failure of the system	35
Figure 25	Graphic of the amount of money spent on the project.	38
Figure 26	Connection between the operator <code>addMessage</code> and an Agent in Touch Designer	42
Figure 27	Connection between the operator <code>agentSession</code> and an Agent in Touch Designer	42
Figure 28	Workflow for the connection of personal documentation to make the agent learn based in the LLM	43
Figure 29	External tools connected to an agent	44
Figure 30	Parameters of the RAG_index operator	45
Figure 31	Error creating index in RAG_index	45
Figure 32	Structure inside of the RAG_index .	47
Figure 33	Lines added to fix the error.	48

LISTINGS

1	Python code to clean problematic characters	49
2	Python code to do the embeddings	50

ACRONYMS

[LLM](#) Large Lenguaje Model

AI	Artificial Intelligence
VHS	Video Home System
VJ	Video Jockey
VDMX	Video Digital Media eXchange
3D	3 Dimensions
TOP	Texture Operator
CHOP	Channel Operator
SOP	Surface Operator
DAT	Data Operator
COMP	Components
LOPS	Language / Learning / Logic / Latent Operators
RAG	Retrieval Augmented Generation
MCP	Model Context Protocol
HTML	HyperText Markup Language
CSV	Comma Separated Values
ID	Identity
TXT	Plain Text Archive
API	Application Programming Interfaces
OP	Operator
GLSL	OpenGL Shading Language
PDF	Portable Document Format
TCP	Transmission Control Protocol

INTRODUCTION

1.1 MOTIVATION AND RESEARCH PROBLEM

This project was motivated by a lot of academic interests and personal experiences. A strong curiosity for both audio and video has always been shared by us. Coming from backgrounds in audiovisual engineering and graphic design, we tend to approach creations with a critical and analytical eye. When it came time to choose a topic for our final master's project, we realized quickly that our interests were closely aligned. After engaging in numerous in-depth dialogues, we arrived at the decision to concentrate our research into two primary areas: the realm of visuals and music, along with the dynamic interplay between these two forms of expression. This intersection felt like the natural place where our skills, passions, and professional perspectives converged.

As we continued our research, we observed that in live shows, the visuals often had little or no meaningful connection to the music. Many visual productions are undeniably impressive. The work of Anyma [1] is an example of this. However, these productions tend to function primarily as aesthetic elements. Rather than enhancing the auditory experience or accentuating the distinctive attributes of each composition, the visual elements often prove to be a source of distraction.

We also discovered that a significant number of artists rely heavily on tutorials for creating their visuals. Tools like TouchDesigner, widely used in the industry, offer powerful capabilities for generating dynamic visual content. The platform offers numerous step-by-step tutorials [6] aimed at beginners, guiding them through the creation of basic visual effects. However, we were surprised to find that even at major, internationally recognized festivals, some artists were using visuals that were essentially the same basic tutorial outputs, sometimes with only minimal modifications, as shown in Figure 1. This highlighted a gap between the potential of the technology and the creative depth actually being applied in many professional contexts.

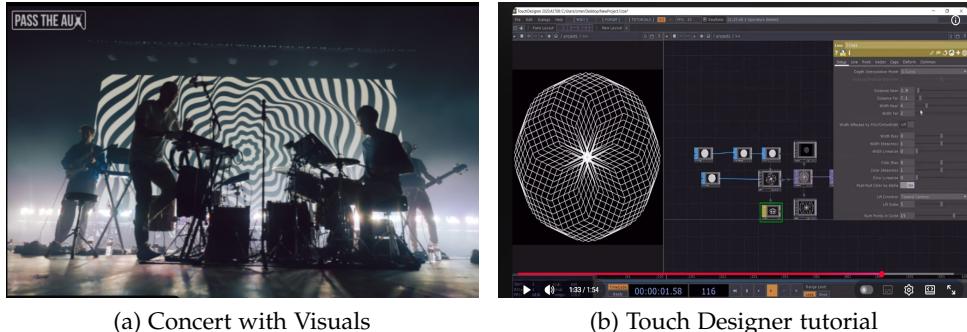


Figure 1: Similarities between visuals in concerts and Touch Designer tutorials, focused on simple geometrical shapes with audioreactivity

1.2 HYPOTHESIS AND OBJECTIVES

The issues surrounding the current use of visuals were recognized, and it was decided that this challenge would be made the foundation of our thesis. Since TouchDesigner has been a key tool throughout the Master's program and is widely regarded as a primary platform for creating visuals, we formulated the following research question: **Is it possible to develop a system within TouchDesigner that helps artists create unique visuals more easily?**

The rapid growth of artificial intelligence and the increasing accessibility of **LLM** based tools, such as Microsoft Copilot and Cursor, are giving rise to new workflows that integrate Artificial Intelligence (**AI**) directly into creative and professional environments. These advancements indicate that **AI** has the potential to play a significant role in supporting or even improving digital creation processes. Our concept is to incorporate artificial intelligence into TouchDesigner, empowering users to customize its training according to their unique requirements and artistic inclinations. This would enable artists to generate unique visuals more efficiently and reduce their dependence on generic tutorials, encouraging more personalized and innovative visual production. Based on that hypothesis, the next objectives were proposed to accomplish the project:

- **O₁ Research and training:** The first objective is to establish a solid research foundation for training **LLM** based

agents. The tool is intended to be customizable so that each artist can adapt it to their own creative workflow. This initial stage will involve selecting a specific artist as a reference case. The goal is to identify a viable method for training an **LLM** based model and subsequently integrating it into TouchDesigner. This requires selecting an appropriate model, determining how to structure and transform information so that it can be effectively learned by the model, and implementing a workflow that enables seamless integration within the TouchDesigner environment. Achieving this will ensure that the system can operate reliably and support the intended creative processes.

- **O₂ Creation of the tool:** The second objective focuses on developing the tool within TouchDesigner. This stage involves building a network of **AI**-driven agents that are trained using the research compiled in the first objective. The goal is for this network to support a wide range of creative tasks, such as modifying colors and shapes, generating shaders, and producing audio-reactive visuals. Users should be able to accomplish these tasks by either providing an example visual or entering a prompt. In the end, this system tries to make the creative workflow more efficient and give artists the ability to create more personalized, expressive visuals more easily.

2

THEORETICAL FOUNDATIONS

2.1 VISUAL HISTORY

The world of screen visuals is a rapidly evolving field that brings together art, technology, and design. Far from being limited to the simple reproduction of images, this discipline encompasses a wide range of techniques, styles, and audiovisual tools. It includes everything from digital manipulation and real-time rendering to generative art and immersive installations. Today, visuals play a central role in technological art, contributing not only to aesthetic expression but also to musical and interactive experiences.

The origins of this practice can be traced back to the 1960s with the introduction of Sony's Portapak, the first portable analog video recorder, shown in figure 2. This device made it possible for creators to experiment with video outside of traditional television studios, opening the door to new forms of artistic expression. Among the first to explore its possibilities was Nam June Paik used magnets and electromagnetic filters to distort electronic images and create innovative visual effects [8, 18]. At the same time, artists such as Steina and Woody Vasulka focused on developing analog synthesizers capable of modulating waves and altering electronic signals in real time [22]. Although these early explorations often intersected with scientific experimentation, they laid the foundation for what would eventually become a recognized artistic field.



Figure 2: Sony Portapak camera (Maison de la Vidéo & du Cinéma: [11])

During the 1980s and 1990s, with the advent of digital video, new techniques for editing, production, and temporal manipulation emerged. At this point, artists such as Bill Viola began to explore new techniques using extreme video slow motion and careful compositions [23]. Pipilotti Rist experimented with modifying images by altering saturation and distorting them to create expansive projections [21]. At the same time, in the field of electronic music, Video Jockey (VJ) was invented, a practice that consisted of mixing images in real time while music was playing, with people like Colo Miura [13], who stands out using all kinds of wild analog techniques very early on using flames, water, video feedback, and Video Home System (VHS) to VJ experimentally. Also performing with Boredoms [17], doing visuals while the group was playing. This concept quickly grew with the development of software such as Modul8, Video Digital Media eXchange (VDMX), and Resolume [20].

Starting in the early 2000s, the availability of powerful computers, advanced graphics cards, and high-brightness projectors profoundly transformed visual creation processes. These tools, which were once static, have evolved into dynamic systems capable of generating images in real time, responding to external data, and integrating with sensors or interactive devices. Rather than approaching visual creation as a fixed, pre-defined product, it became possible to use environments such as TouchDesigner,

Max/MSP/Jitter, and Pure Data was used to create a visual experience based on a modular data flow. [4].

Concurrently, languages such as Processing, Open frameworks and p5.js enabled the growth of generative art, empowering algorithms, mathematical frameworks, and computational logic to dictate the structure and behavior of images [15]. Consequently, the role of the visual artist underwent an evolution, becoming a hybrid profile that integrates design competencies with programming acumen and technical experimentation. Visual creators shifted from operating tools to becoming system architects who can manage complex information flows to produce flexible, interactive audiovisual experiences.

Today, screens have expanded far beyond the traditional rectangular device, with new options including curved screens, foldable screens, and screens with various sizes and shapes. These elements can now manifest in various forms, including architectural surfaces, immersive environments, interactive installations, and high-resolution urban displays. Sensors, depth cameras, body-tracking devices, and real-time analysis systems are integrated to allow artworks to react to the presence and actions of the audience, creating multisensory experiences in which the image behaves like a living environment. As Paul [14] notes, contemporary digital art operates in a hybrid space where physical materiality and computational logic converge, thereby transforming the relationship between viewer and artwork.

This landscape is defined by its technical diversity and complexity. A variety of tools and methods from different fields are used to create screen-based visuals. These fields include composition, animation, programming, interaction design, data visualization, digital scenography, and algorithmic systems. Therefore, an interdisciplinary territory is worked in by the contemporary artist, and mastery of both aesthetic strategies and technological capabilities is required. The screen is no longer merely a display surface but an expanded field where human creativity, electronic processes, and computational structures intersect. The way it keeps changing reflects the big impact of new technology on modern art and the way it moves towards using visual practices that go beyond the usual limits of images and exhibition spaces.

2.2 TOUCH DESIGNER

TouchDesigner is a real-time visual development platform developed by Derivative that is used extensively for the creation of interactive multimedia systems, data-driven visualizations, and generative art. The architectural framework of this system is predicated on a node-based procedural workflow, a methodology that empowers creators to construct intricate systems by establishing connections between functional units designated as operators [5]. This modular approach is conducive to iterative design and facilitates the utilization of TouchDesigner by both artists and technically oriented users, aligning with the broader tradition of visual programming tools in new media [9].

A significant advantage of TouchDesigner is its capacity for real-time rendering and data processing. This enables the manipulation of video, 3 Dimensions (3D) geometry, audio, and sensor inputs with minimal delay. The software's capacity to perform such functions has led to its central role in the creation of large-scale audiovisual installations, projection mapping, interactivity in stage design, and immersive experiences [2]. Its real-time nature situates the platform within contemporary practices of live media and performance technologies, where responsiveness and dynamic interaction are essential [3].

The TouchDesigner workflow is organized into specific categories of operator, namely Texture Operator (**TOP**s), Channel Operator (**CHOP**s), Surface Operator (**SOP**s), **DAT**s, and Components (**COMP**s). Each operator category has been designed to process a particular data or execute a specific process. This layered structure enables creators to transition between surface-level interaction design and more profound computational logic [10]. The platform utilizes Python as its scripting environment, providing advanced control, automation, and logic-based behavior for interactive systems [19]. TouchDesigner's integration of visual programming with textual scripting positions it at the nexus of creative coding environments, such as Processing [16], along with live coding paradigms that have been explored within performance contexts [12].

2.3 LOPS

Language / Learning / Logic / Latent Operators (**LOPS**) is a modular AI framework designed to deeply integrate artificial intelligence into TouchDesigner created by Lyell Hintz, aka Dot-Simulate. Its tool ecosystem allows AI agents to reason, search, perceive, remember, and act directly inside node-based visual workflows. Every **LOPS** tool is implemented as a TouchDesigner operator and can expose its functionality to **AI** agents through a standardized tool interface, enabling structured, reliable **AI** driven execution.

At the center of the system is the idea that **AI** is not a black box, but an active participant in the network, able to inspect context, call tools, retrieve information, manipulate data, and respond in real time.

- **TextCore AI Controllers**

The **Agent operator** is the main intelligence hub. It manages conversations, system prompts, context windows, streaming responses, and tool execution. The Agent dynamically decides when to call tools, passes structured arguments to them, and incorporates their results into its reasoning. It supports text, image, and audio context and acts as the orchestrator for all **AI** workflows.

Gemini live extends this functionality into real-time, multimodal interaction. It enables live voice conversations using speech-to-text and text-to-speech, supports continuous streaming input/output, and allows **AI** agents to call tools during spoken conversations. This makes it suitable for installations, performances, and interactive systems.

The **Role creator** tool assists in dynamically generating system prompts and **AI** personas. Instead of manually crafting long instructions, users can generate specialized **AI** roles such as tutor, analyst, creative assistant, which can then be assigned to agents.

- **Tool Management and Context Awareness**

The **Tool Registry** automatically discovers all available tool operators in a project and makes them accessible to agents.

This removes the need for manual configuration in complex networks and allows scalable multi-agent setups.

The **Tool Monitor** provides agents with awareness of user activity inside TouchDesigner. It tracks selected nodes, parameter changes, errors, and interaction events, allowing the [AI](#) to respond intelligently to what the user is doing in real time.

The **Network Context** tool exposes structural information about the TouchDesigner network—operators, connections, and layout so agents can reason about the project itself.

- **Search, Web, and Data Acquisition Tools**

The **Search operator** enables [AI](#) driven queries across multiple search providers and data types. Agents can autonomously perform web searches, retrieve results, and integrate them into their responses.

Source Webscraper crawls websites while respecting rate limits and robots.txt, extracting clean text into tables.

Source Crawl4AI uses a headless browser to render modern websites and export content as structured Markdown, suitable for large-scale crawling.

textSource GitHub ingests repositories, issues, documentation, and code for knowledge extraction.

Source Docs parses local documents (HyperText Markup Language ([HTML](#)), Python, Markdown, etc.) into indexed tables.

Source Ops extracts information directly from Touch Designer networks, turning operators and parameters into searchable knowledge.

The **Save Sources** operator persists scraped or generated content as Markdown files, allowing datasets to be reused or indexed later.

- **Retrieval Augmented Generation and Memory Workflows**

LOPS supports [RAG](#) by allowing content to be indexed and retrieved dynamically. While the indexing and retrieval operators work behind the scenes, the overall workflow enables agents to pull relevant information from large datasets

instead of relying solely on their prompt context. This allows scalable knowledge bases, documentation assistants, and project-aware AI systems.

- **Integration and External Tooling**

The **MCP Client** connects TouchDesigner to external tools using the **MCP**. This allows agents inside TouchDesigner to call tools hosted outside the application, extending **LOPS** beyond local functionality and enabling distributed **AI** systems. When TouchDesigner also runs an **MCP Server**, external agents can call tools exposed by Touch Designer itself, allowing direct control of parameters, operators, and systems inside Touch Designer. This makes **MCP** a bidirectional bridge, enabling Touch Designer to both orchestrate external **AI** tools and be orchestrated by outside agents such as Cursor, Blender, or other **MCP**-enabled environments.

- **Utility and Data Manipulation Tools**

Several tools focus on enabling **AI**-controlled manipulation of Touch Designer data:

Tool DAT allows agents to read, write, and modify table data programmatically.

Tool Parameter lets agents change operator parameters in a structured, validated way.

The **Super Select** operator enhances table manipulation with advanced filtering and fuzzy selection, making it useful for preparing data before **AI** processing

- **Environment and Dependency Management**

The Python Manager automatically manages Python virtual environments and dependencies required by **LOPS** operators. This ensures stability and reduces setup friction, allowing complex **AI** pipelines to run reliably inside Touch Designer

All of this information, along with usage examples, installation tutorials, recommendations, and additional technical details, is available in the official **LOPS** documentation [7].

2.4 LARGE LANGUAGE MODELS (LLM)

A **LLM** is an artificial intelligence system designed to communicate with humans. These systems can understand, generate, and reason with human language. Essentially, it reads inserted text and predicts what text will come next based on it. By repeating this process, the system produces coherent language. These systems essentially seek to mimic human reasoning by learning statistical patterns from large amounts of text.

These models perform a task called next token prediction. For instance, when the phrase "The capital of France is" is entered into the system, the model predicts "Paris." This interaction creates a token, which can be a word, part of a word, a punctuation mark, and so on. After performing these processes repeatedly, the system acquires more complex behaviors, such as writing essays, answering questions, translating, and writing code. Before the text can be processed by the model, it must first be converted into tokens. Each syllable or word is converted into a token and assigned a numerical identifier. Approximately 30,000-100,000 tokens are needed to translate a text. This mapping of tokens can be imagined as large matrices containing lists of numbers. The more similar the words are, the more similar their vectors will be. This process is called "embedding." With each interaction, the model uses a process called "self-attention," which involves looking at all previous tokens to decide which ones matter most for predicting the next one. In this way, weights are assigned to previous tokens based on their relevance. Rather than using just one attention mechanism, these models use many in parallel, each responsible for a specific topic. This approach achieves better understanding and greater linguistic richness. After applying these weights to each token, each token goes through a neural network to introduce nonlinearity and abstraction. This enables the model to learn more complex concepts.

These models are designed to be trained using large amounts of data, such as books, articles, web pages, and repositories. Artists using the tool developed in this project can use it to train the model with all their research, color patterns, shapes, and any relevant files. **LLMs** learn by masking text and predicting missing or future tokens while minimizing prediction errors. After pre-training, models are refined using examples of specific com-

mands, human feedback, or counterfactual datasets. Once the model has been trained and refined, it can generate text based on a user-generated prompt. As the user types, the prompt is converted into tokens that pass through the model. The probabilities of the next token are then calculated and the token selected using a sampling strategy.

One of the characteristics of these models is that they do not think like humans or act logically. They learn patterns and imitate logical steps based on their training, making them very powerful tools for solving multi-step problems or generating analogies. However, they will fail if asked to do something outside their training parameters. This can result in the generation of false but plausible information, as well as a false sense of security due to invented sources. As they have no consciousness or grounded experience, they can only remember what they have previously seen and are incapable of retaining memory. In other words, they cannot retain past information in order to hold a conversation with the user on their own.

3

RESEARCH

3.1 INTRODUCTION

For the practical part of the project, the focus will be on accomplishing all the objectives. The proposed approach is to first investigate and define an effective method for training the model, and only then proceed with the development of the tool itself. Since the project is intended for use by artists, a reference artist with a well-defined visual identity, particularly in terms of color palette and formal language, was selected. This choice makes it possible to clearly demonstrate how the system generates visuals that are informed by and aligned with the artistic characteristics of the chosen individual. This will allow each artist to adapt and modify it according to their own creative needs. The development process will center on a single case study. To simulate an artistic profile, the decision was made to use the color palettes of Antoni Gaudí.

Gaudí was a renowned Catalan architect whose work is characterized by organic, curved forms that evoke elements of religion and nature. His architectural language combined structural innovation with strong symbolic and aesthetic intent. One of the most distinctive techniques associated with Gaudí is trencadís, a modernist method that involves covering surfaces with fragments of broken ceramic tiles. As seen in figure 3, this technique allowed for the creation of vibrant, textured surfaces and complex color compositions, which have become a defining feature of his work. The use of Gaudí's color palettes and formal principles provides a clear and recognizable visual reference, making it well suited for simulating an artist-driven visual system within the project.



Figure 3: Images of Gaudí’s Architecture

This material will serve as the foundation for training our system. By grounding the tool in the specific characteristics of his style, we aim to create a version of the project that reflects his artistic identity and demonstrates how the system can be tailored to other artists.

3.2 TRAINING OF THE LLM

In order to properly train the system, information on several different topics will be collected to carry out the entire research process. Several approaches were explored in order to train the **LLM** models. While the objective was clearly defined, to integrate a trained model into Touch Designer and enable interaction directly within the application, the specific method for achieving this integration was not initially known. As a result, different strategies were investigated and evaluated to determine the most effective way to train the model and connect it to the TouchDesigner environment.

3.2.1 First approach

The first approach involved manually creating all the information required for the LLM models. As explained in the 2.4 section, for a model to acquire and use information, it must pass through several distinct stages. First, a document containing the relevant and curated information must be created. Next, this data must be embedded so that the model can process and analyze it effectively. Finally, the model must be connected to the processed data, enabling it to retrieve and use the appropriate information in response to user queries. The first thing was to recollect all the information that the model has to learn. To start researching, as one of the aspects that the project wants to cover was the colour, a research of academic papers based on colour started. After several hours of analysis, approximately 60–70 papers were selected for each topic, prioritizing those that offered the most reliable and impactful insights for the tool’s development.

Once the papers had been selected, the next step was to construct the database. To organize the information systematically, a CSV file was created to store and classify the key details from each article. The following structure, shown in Figure 4 was chosen:

- **Title:** This section contains the title of the paper.
- **Author:** This section contains the authors who write the article.
- **Year:** This section contains the year when the article was published.
- **Summary:** This section contains a condensed version of the paper’s abstract. Because abstracts are typically accessible for free, they provide enough information for the system to understand the essence of each study without requiring full-text access.
- **Keywords:** Keywords were included to enable the system to retrieve the most relevant papers based on the prompts provided by the user. When embeddings are generated, these keywords help the system match user queries with the sci-

entific articles that best fit the topic, ensuring accurate and efficient information retrieval.

This structured approach ensures that the agent can navigate the database effectively and rely on well-organized, high-quality scientific data.

1	hors,year,summary,keywords
2	"Gegenfurtner, K. R. & Kiper, D. C.", 2003, "This paper reviews the neural mechanisms underlying color perception in humans. It describes how the retina encodes color
3	demonstration of functional specialization in human visual cortex", Zeki, S., 1991, "this study demonstrates that the human visual cortex is organized functionally s
4	equential stages of colour processing in the human brain", Zeki, S., 1998, "This paper describes three cortical stages in color processing in the human brain. The first occur
5	s in the color center in the human visual cortex", Bartels, A. & Zeki, S., 2006, "This study analyzes the functional organization of area V4, considered the "color
6	center of color in the brain", "Birds, M. A. et al.", 2014, "This study investigates how the brain encodes colors into perceptual categories. Using fMRI, the authors fo
7	cus on color in the human visual cortex; an fMRI review", "Mallin, V. T.", 2019, "This review analyzes fMRI studies on how different regions of the human visual cortex encode
8	activity patterns predict behavioral performance", "Bartels, M. M. & Bartels, A.", 2018, "This study analyzes fMRI studies that neural activity in the V4 area of the human brain predicts p
9	g the effect of red and blue on cognitive task performance", "Xia, T. et al.", 2016, "This study analyzes how the colors red and blue influence cognitive performance. The re
10	ssessing of color preference in the brain", "Rao, A. W. et al.", 2019, "This study investigates how the brain processes color preferences and which neural areas are involved, o
11	neified theory of visual area V4", "Rao, A. W. et al.", 2012, "This paper proposes a unified theory on the function of the V4 visual area, integrating findings on color, sh
12	ation of perceived object shape by the human lateral occipital complex", "Kourtzi, Z. & Kanwisher, N.", 2003, "This study demonstrates that the lateral occipital complex (l
13	evel occipital complex and its role in object recognition", "Geil-Spector, K., Kourtzi, Z. & Kanwisher, N.", 2001, "This study delves into the role of the lateral occipital
14	ception reduces activity in human primary visual cortex", "Murray, S. O. et al.", 2002, "This study reveals that coherent perception of form reduces activity in the primary
15	p-scale organization of shape processing in visual cortex", "Freud, E. et al.", 2017, "This study explores the large-scale organization of shape processing in the human visi
16	g material versus geometric properties of objects", "Neeman, S. D. et al.", 2005, "This study examines how the brain differentiates between imagining the material properties
17	of visually evoked potentials: paradigms and perspectives", "Vialatte, F.-B., Maurice, M., Daubens, J. & Cichocki, A.", 2010, "This article provides a comprehensive review
18	of steady-state visual evoked potential in vision research", "Noria, A. M. et al.", 2015, "This review analyzes the use of steady-state visual evoked potentials (SSVEPs) in vision
19	istics of steady-state and transient responses evoked by modulated light", "Regan, D.", 1966, "This classic study analyzes the brain's visual responses to modulated light s
20	6 responses to 1-100 Hz flicker: resonant phenomena in visual cortex", "Hermann, C. S.", 2001, "This study examines how the human brain responds to intermittent visual sti
21	: of strong harmonics during visual entrainment", "Heinrichs-Graham, E. et al.", 2012, "This study investigates the generation of strong harmonics in the visual cortex during
22	gamma-band entrainment of visual cortex", "Nietro, N. M. et al.", 2024, "This study explores how the human visual cortex responds to entrainment in the gamma band (~30-100 Hz)
23	frequency entrainment and r Col2: author", "Chow, C. et al.", 2016, "This study analyzes how rods (low-light-sensitive photoreceptors) contribute to cortical entrainment an
24	ed fast SSVEPs during sleep", "Sharon, O. et al.", 2018, "This study investigates how steady-state visual evoked potentials (SSVEPs) are affected during sleep. Using EEG, th
25	stimulation under 4 Hz, not 10 Hz, generates distinct responses", "Retter, T. L. et al.", 2025, "This study analyzes how different frequencies of visual stimulation affect co
26	flicker fusion frequency: a narrative review", "Mankowska, N. D. et al.", 2021, "This narrative review analyzes the critical flicker fusion frequency (CFF), which is the sp
27	eeval crossmodal pitch/vision associations", "Sciortino, P. et al.", 2023, "This study explores crossmodal associations between sound [pitch] and visual perception using ste
28	combined with behavioural tasks in perception research", "Wittenhagen, L. et al.", 2019, "This study combines steady-state visual evoked potentials (SSVEPs) with behavioral t
29	f computational neuroaesthetics", "Li, R. et al.", 2020, "This review analyzes the emerging field of computational neuroaesthetics, which combines neuroscience, psychology,
30	ive scope review of neuroaesthetics and visual perception", "Atiyeh, B. et al.", 2025, "This narrative review addresses neuroaesthetics and visual perception, exploring ho
31	and reshaping the aesthetic brain: emerging perspectives", "Kirsch, L. P. et al.", 2016, "This article reviews how aesthetic experience modifies brain activity and visual pe
32	ctrical brain imaging on perception and appreciation of pictures", "Maglione, A. G. et al.", 2017, "This study uses electrical neuroimaging (EEG) techniques to investigate h
33	color and luminance from responses in human visual cortex", "Bartels, M. M. & Bartels, A.", 2013, "This study investigates how the human visual cortex encodes color and lum
34	otion, and attention in the human visual cortex", "Bartels, A. et al.", 2008, "This study explores how the human brain simultaneously processes color, motion, and attention
35	tribution of area V4 to human color vision", "McKeefry, D. J. & Zeki, S.", 1997, "This study analyzes the role of the V4 area in color perception in humans. Using fMRI and le
36	tched neural systems for shape perception", "Kourtzi, Z. et al.", 2003, "This study investigates how the brain perceives shapes using a distributed neural network that include
37	rial EEG analysis reveals burst structure during flicker stimulation", "Oppermann, H. et al.", 2024, "This study uses EEG analysis in individual trials to examine the struc
38	ture design affects harmonic content of SSVEP responses", "Solf, B. et al.", 2020, "This study investigates how the design of visual stimuli influences the harmonic content of S
39	perspectives on gamma-band visual entrainment", "Duercker, K. et al.", 2021, "This article critically reviews studies on visual entrainment in the gamma band (~30-100 Hz).

Figure 4: Example of a [CSV](#) with the structure of the project

The next step involved processing the information contained in the [CSV](#) file. To ensure that the agent could generate embeddings effectively, it was necessary to remove any characters that might interfere with processing, such as accents, apostrophes, and other special symbols. These characters can cause inconsistencies or errors during embedding generation, so cleaning the data was essential.

To accomplish this, a script was created to automatically convert and sanitize the [CSV](#) files, ensuring that all entries were standardized and compatible with the agent's requirements. This script, titled `Papers_clean.py`, performs the necessary preprocessing steps and prepares the dataset for seamless integration. The full code for this script is included in the appendix [B.1](#).

The next step was to create the embeddings. To do this, as we had some knowledge of Python but not of how to create embeddings, we used Cursor. Cursor is paid software similar to Microsoft Copilot, which allows you to program in an environment that has implemented artificial intelligence that allows you

to read and analyze everything you are doing, as well as suggest possible improvements.

Basically, this code builds and tests a semantic search system for academic papers. It loads a cleaned [CSV](#) file containing paper data and ensures required text fields exist. Each paper's title, summary, and keywords are merged into a single text string. A pretrained SentenceTransformer model converts these texts into numerical embeddings. The embeddings, original texts, and metadata are stored in a persistent ChromaDB vector database. Each document is assigned a unique Identity ([ID](#)) and saved to disk for reuse. A test query in natural language is also converted into an embedding. The database is searched for the most semantically similar papers to the test query. The top matching documents are retrieved based on meaning, not keywords. Finally, the script prints the titles, metadata, and text snippets of the best results. For watching the full code, go to the annex [B.2](#)

The subsequent step was to integrate the generated embeddings into Touch Designer. However, during one of the meetings, the possibility of working with [LOPS](#) was discussed. [LOPS](#) is a Touch Designer-based system that enables the integration of [LLM](#)-powered agents directly within the software.

3.2.2 *LOPS*

When we started working with [LOPS](#), it quickly became clear that it wouldn't be plain sailing. Due to incompatible versions of other software, such as Python, it took two weeks of errors just to install the tool within Touch Designer. Nevertheless, thanks to some helpful tutorials and a Discord server with other people using [LOPS](#), we managed to start working with the software. Once [LOPS](#) had been correctly installed, the research began and a much simpler way of training the model was discovered than that described in the [3.2.1](#) subsection. Training the [LLM](#) model involves fewer steps, and [LOPS](#) automates the process much more efficiently. However, before starting, two Gemini 2.0 Application Programming Interfaces ([API](#)) keys had to be acquired, one for each team member. Ollama, an [LLM](#) model designed for local use, also had to be installed. This model enables the creation of embeddings. Below, we will explain how to train an [LLM](#) model and connect it to an agent, enabling you to access information based on a prompt.

- **Source Docs:** The first step is to incorporate all user-provided information into Touch Designer. This is achieved using an operator called Source Docs. This operator enables users to reference a folder on their local computer and automatically read all documents stored in supported formats, such as .html and .py. These files collectively form the knowledge base that the agent will later use.

When the operator is opened in Touch Designer, the configuration interface becomes visible, as seen in Figure 5. From here, users can define the folder path and manage how documents are ingested into the system. This ensures that all relevant information is correctly loaded and made available for further processing.

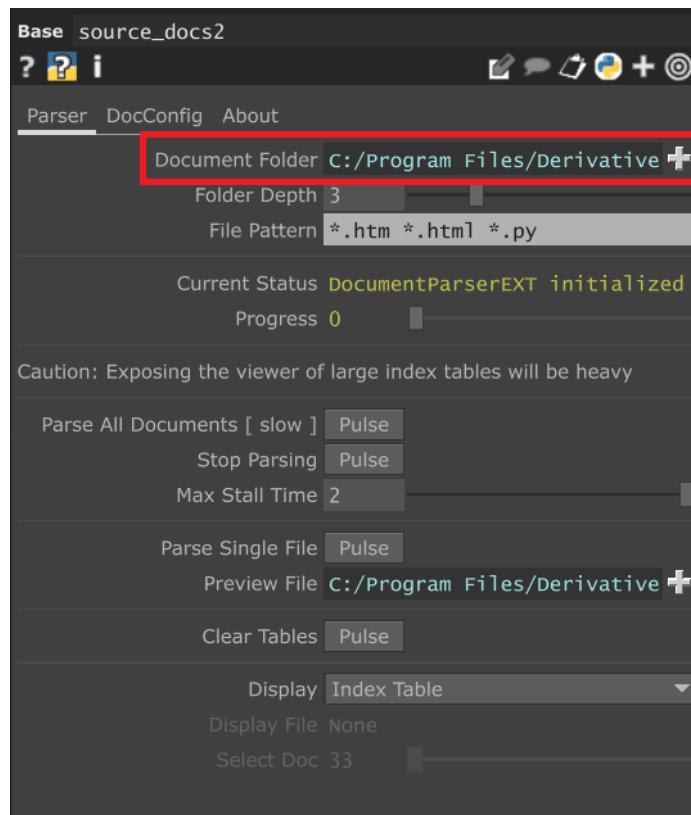


Figure 5: Place to select the Folder with all the documents to be embedded by the LLM

Once the folder has been selected, the documents must be parsed so that the system can correctly process their contents. The operator interface provides three main options for this, as shown in Figure 6.

The first option, 'Parse All Documents' (slow), processes all the documents in the selected folder in one operation. This method is useful when working with a complete dataset, although it may take longer depending on the number and size of the files. The second option, 'Parse Single File', processes each document individually, offering greater control when testing or updating specific files. Finally, the 'Clear Table' function removes all previously parsed data. This option is necessary when a document in the folder has been modified, as the existing data must be cleared and re-parsed to accurately reflect the changes.

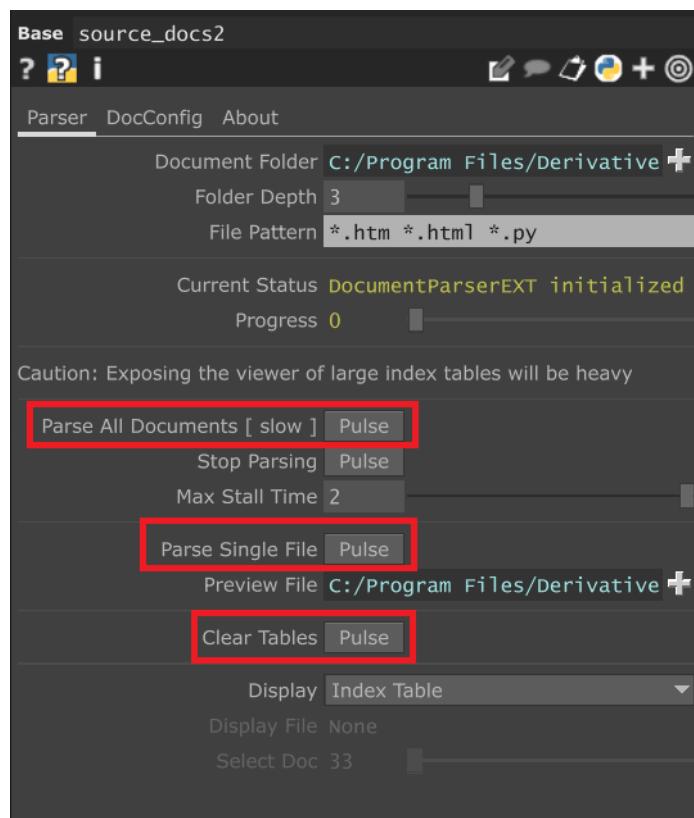


Figure 6: Options Parse All Documents, Parse Single File and Clear Tables

- **RAG index:** When all the documents in the folder that the user wants have been parsed, the next step, as shown in Figure 7, is to connect the 'RAG index'.

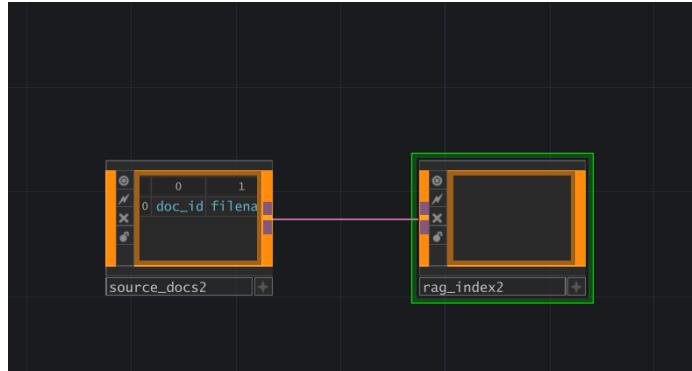


Figure 7: Connection between Source Docs and [RAG](#) index

This operator is responsible for processing all the documents provided by the Source Docs operator and generating the corresponding embeddings. As with the previous operator, the [RAG](#) Index includes a drop-down menu that provides access to its configuration options. One of its key functions is allowing the user to select the [LLM](#) model that will be used for embedding generation and retrieval.

In this project, a local instance of the Llama 3 model was used, running through Ollama. Once the model is selected, the operator processes the ingested documents, creates embeddings from the parsed content, and builds the corresponding indexes that enable efficient information retrieval. These functionalities and configuration options are illustrated in Figure 8. Depending on each person’s computer, this step is prone to errors. If you encounter an error at this stage, please refer to the appendix [A.3](#).

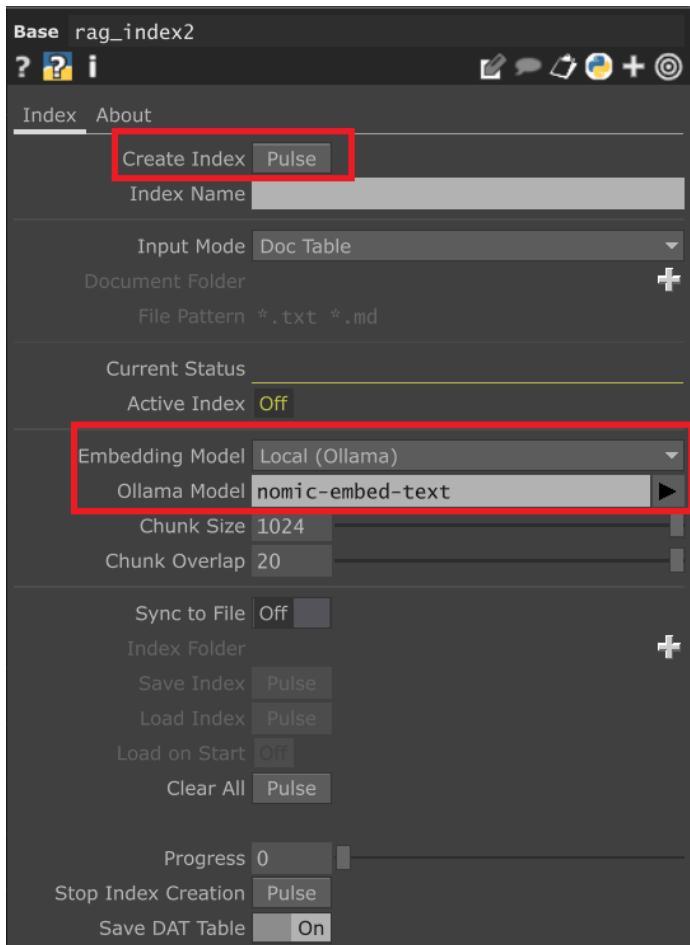


Figure 8: Options of the RAG index

Once the index is created, the index name will appear automatically. To verify everything is correct, an image like the Figure 9 should appear.

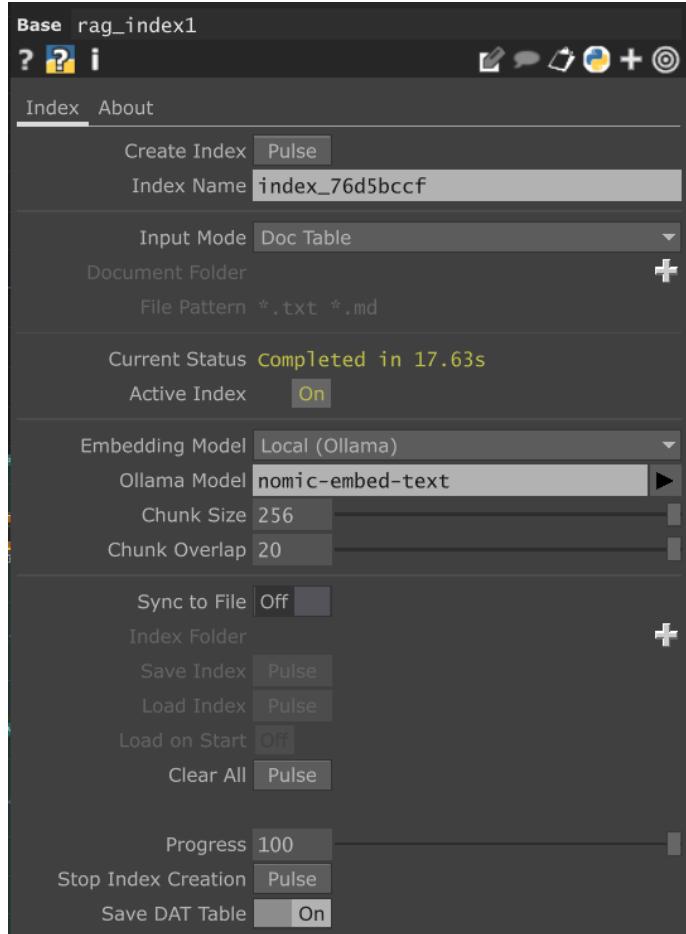


Figure 9: Index created

- **RAG Retriever:** Once the **RAG** Index has been created, the next step is to add a **RAG** Retriever operator. This operator must be linked to the previously generated **RAG** index in order to retrieve relevant information during user interactions.

As shown in Figure 10, this configuration is performed through the operator's parameters. In the Search Mode section, the Custom option must be selected. After enabling this mode, the desired **RAG** index should be dragged into the Query Phrase section. Once the **RAG** components are correctly connected, the final step is to click the "Query Index" button. When this action is performed, the operator searches the indexed data and retrieves the most relevant document fragments based on the information requested by the agent or the message entered in the Add Text section. These retrieved fragments are then used to provide context-aware

responses, enabling the agent to answer queries accurately using the embedded knowledge base, as shown in Figure 11.

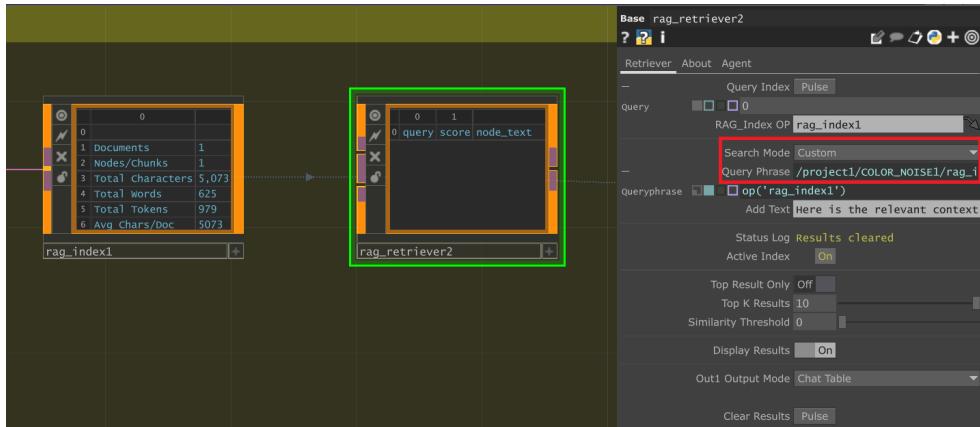


Figure 10: Connecting the **RAG** index into the **RAG** retriever.

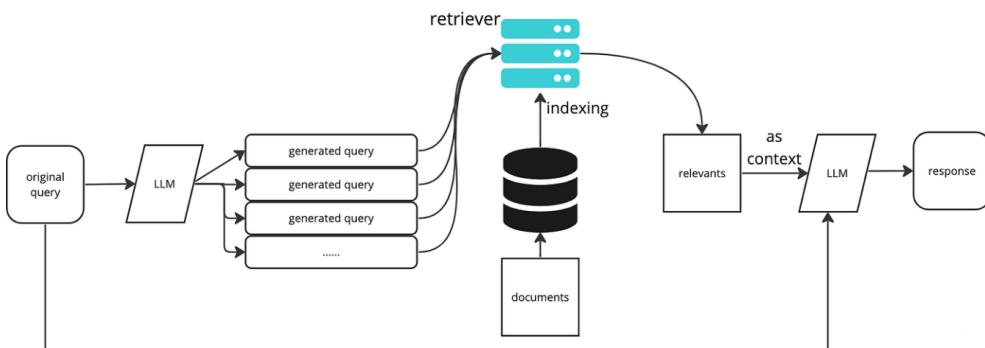


Figure 11: Diagram of how a Multi-Query Retrieval works

- **Agent:** Finally, an agent must be added and connected to the **RAG** Retriever. This enables users to interact with the system directly via the agent, which responds by retrieving and using the most relevant information based on user requests and knowledge stored in indexed documents.

To establish this connection, the agent's parameters must be configured accordingly. As shown in Figure 12, navigate to the 'Tools' tab and locate the 'External Tools Operator (OP)' section. Enable an external operator and drag the corresponding **RAG** retriever into the designated field.

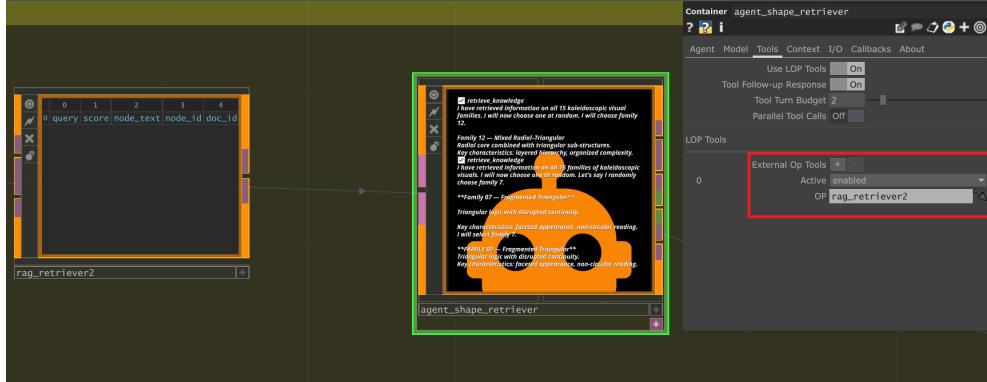


Figure 12: Adding the [RAG](#) retriever into the agent.

Finally, once all operators have been properly connected, the agent must be configured to define its role and expected behavior. This is done by accessing the agent's parameters and specifying its system instructions.

To do so, navigate to the Agent tab and click the EditSysMess button. Within this field, the user can describe the agent's function, define its scope, and specify how it should respond to queries.

3.3 TOOL CREATION

After determining that the project could be effectively developed using [LOPS](#), as described in the previous section, the focus shifted toward designing a workflow that would allow users to upload their own files and generate visuals based on that material. A key objective of this workflow was to preserve a certain level of autonomy within the system. This means that even when the same prompt is entered multiple times, the system is capable of producing different visual outputs, rather than repeating identical results.

The concept is that the system can be trained using a large dataset derived from the user's own research into visual and auditory elements such as shapes, colors, and sounds. This dataset serves as the creative foundation from which the system operates. For the purposes of this project and to validate the workflow, a simplified approach was adopted by working with a limited set of [HTML](#) files specifically focused on the work and color palettes of Gaudí.

To properly structure the tool's workflow, several key components are required. First, the entire database must be imported into Touch Designer, allowing the model to be trained and to internalize the provided information. Next, the system must be capable of extracting relevant data from the stored content in response to a user-defined prompt. Once this information has been retrieved, the system must generate a highly detailed prompt in the form of executable code. This code is then used to drive the creation of shaders through a **TOP** operator known as **GLSL Multi**, enabling the automatic generation of visuals based on the interpreted data.

This workflow, shown in Figure 13, establishes a flexible and scalable framework in which user-provided research material directly informs the visual output, while still allowing for variation, experimentation, and creative unpredictability within the system.

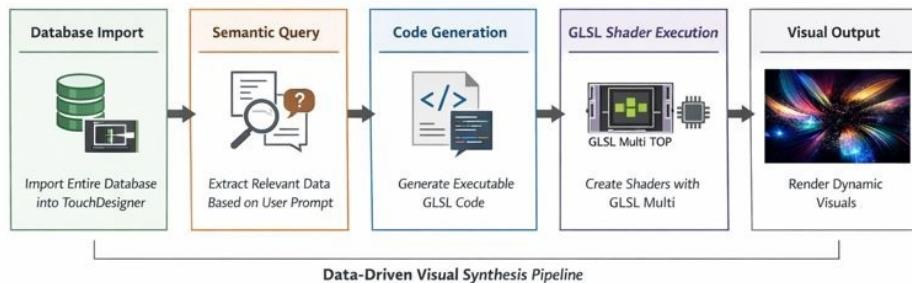


Figure 13: Diagram of the system's workflow.

3.3.1 Orquestration

The most important feature of the tool is its autonomy. A series of operators will perform different functions simultaneously, so there must be a chain of command to coordinate each element.

Within **LOPS**, there is an operator called the 'Agent Orchestrator'. This operator is responsible for managing all the agents that the user requires. As shown in Figure 14, this operator allows you to add as many agents as required for orchestration. Simply access the Orchestrator tab, enter the number of agents you wish to control and drag them into the operator. There is also a button to activate or deactivate any agent without disconnecting it.

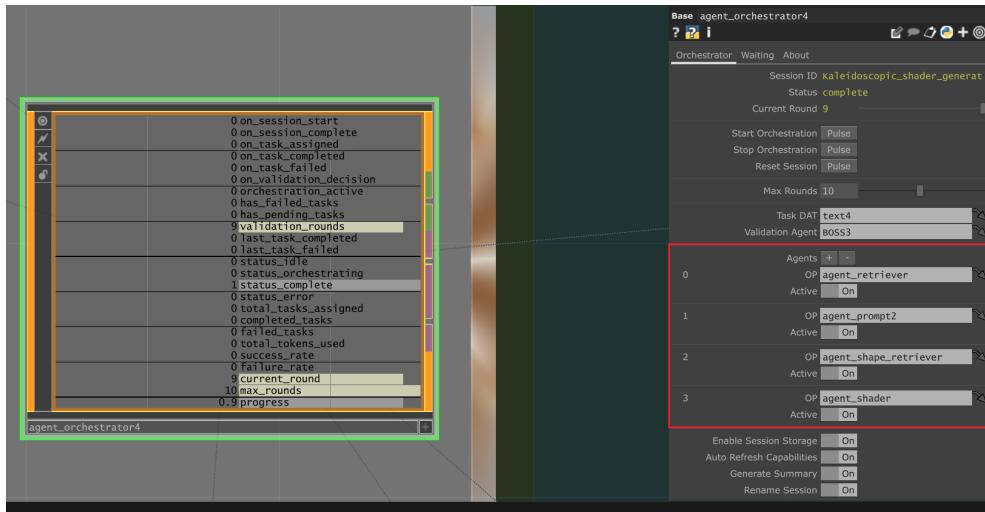


Figure 14: How to add Agents to the Agent Orquestrator.

Once the agents participating in the orchestration have been added, two additional components are required: a prompt containing written instructions, and an agent responsible for coordinating the other agents' actions. As the Agent Orchestrator does not directly communicate with the remaining agents on its own, it requires a standard agent to act as a leader. Instead, the Orchestrator forwards relevant information to the Validator agent, who then distributes tasks and instructions to the other agents involved in the workflow.

In parallel, the orchestration instructions are provided to the Orchestrator in textual form. In TouchDesigner, this is achieved using a **DAT** operator called Text **DAT**, which functions similarly to a basic text editor. This operator enables users to define the orchestration process by writing the sequence of steps and commands. These instructions outline the order of operations and specify the actions that each agent is expected to perform.

Once the leader agent and instruction prompt have been configured correctly, they must be connected to the Agent Orchestrator, as shown in Figure 15.

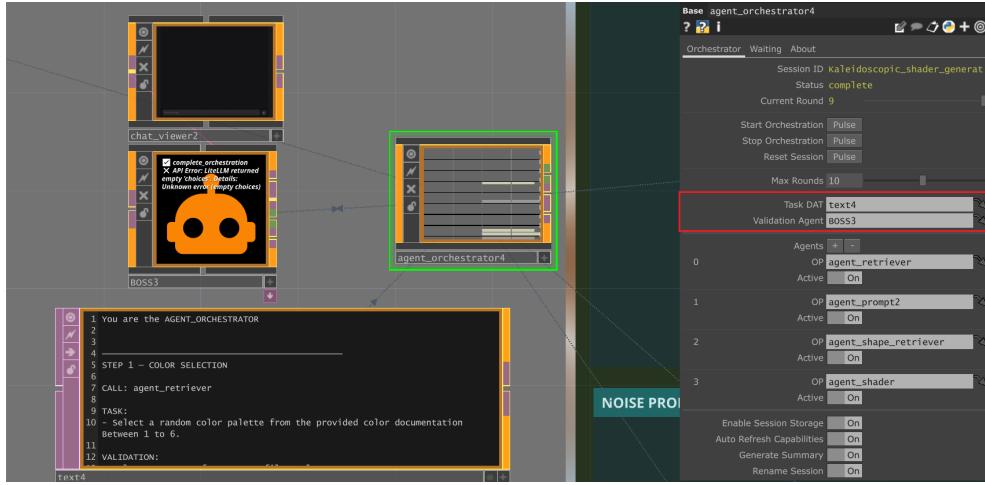


Figure 15: How to add the validator agent and the Task DAT.

This Orchestrator has additional parameters that enable you to start, stop or reset the orchestration, and select the maximum number of rounds to run, as shown in Figure 16.

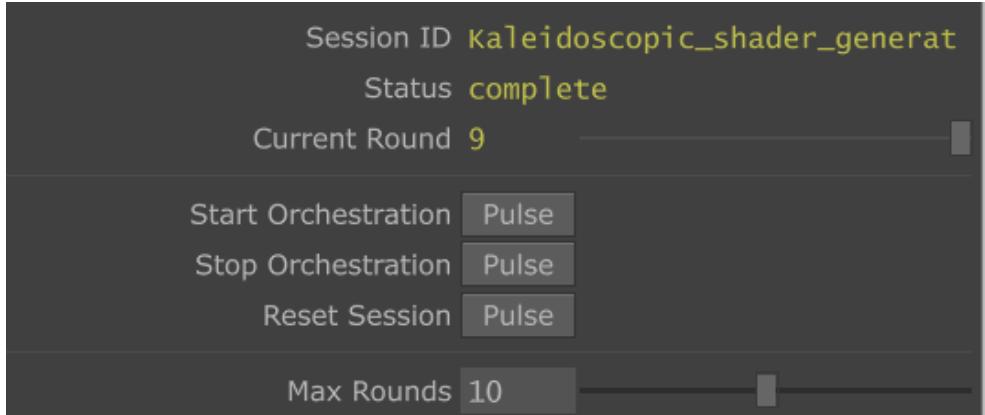


Figure 16: Extra options of the agent orquestrator.

3.3.2 Database Import & semantic query

The following section addresses the topic of data import. Since the complete process, from creating user data to connecting it with the corresponding agent, has already been described in subsection 3.2.2, this part focuses specifically on data structuring and performance optimization.

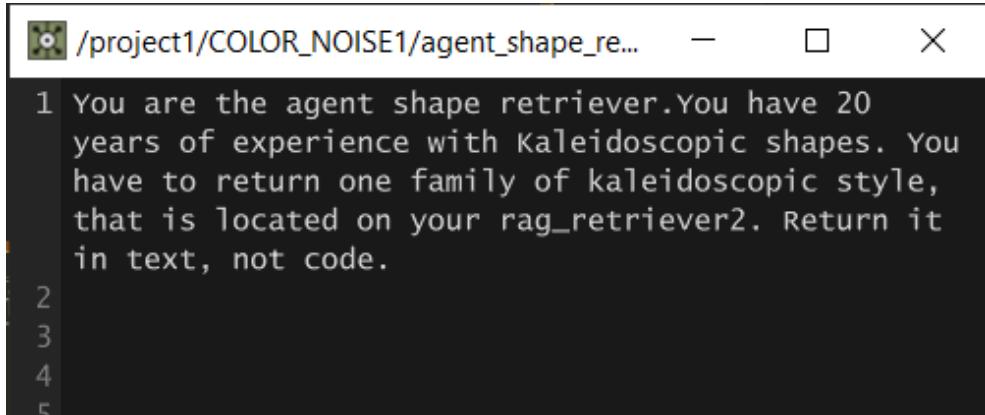
One of the primary considerations is how to structure the information effectively. Through different projects made in testing, it was observed that system performance improves as the information is more finely segmented. In practice, this means that dividing the dataset into smaller, more focused units leads to better results. Rather than relying on a single, generalized agent, it is more effective to deploy multiple agents, each dedicated to a specific task and trained to a high level of specialization.

In the context of this project, experiments were conducted using data related to color and shape. Consequently, each base was supported by two specialized agents, one focusing on color related information and the other on shape related characteristics. The color data used in this project is derived from color palettes associated with Gaudí, drawing inspiration from patterns and combinations found in nature. These palettes serve as a visual reference that reflects organic aesthetics and rich chromatic variation. In contrast, the approach to shapes follows a different strategy. Rather than referencing a single artistic source, the focus is placed on the various types of shaders that can be generated, such as point-based shaders or kaleidoscopic forms. To support this approach, a dedicated database is created for each shader category, with a specialized agent assigned to each one.

Each agent is assigned a specific function that defines the task it is responsible for performing within the system. In addition to connecting agents that manage different data sources, each agent includes an internal configuration section where its role can be explicitly defined. This role specification is essential for ensuring that each agent operates within its intended scope and contributes effectively to the overall workflow.

The following example illustrates how to configure an agent dedicated to generating kaleidoscopic shapes based on an .html file containing visual patterns and instructions for their construction. To define this behavior, the user must access the Agent tab of the corresponding agent and click the EditSysMess button. This action opens an interface similar to the one shown in Figure 17, where the system message can be written to clearly describe the agent's responsibilities and expected output. It is important to ensure that the instructions assigned to the agent do not overlap with those defined in the Text DAT used by the Agent Orchestrator. While the orchestrator provides high-level

coordination and task sequencing, each agent's system message should focus solely on its specialized function.



The screenshot shows a TouchDesigner interface with a dark background. At the top, there is a header bar with a file icon, the path '/project1/COLOR_NOISE1/agent_shape_re...', and standard window control buttons (minimize, maximize, close). Below the header is a text DAT operator. The text inside the operator is organized into numbered steps:

```

1 You are the agent shape retriever. You have 20
years of experience with Kaleidoscopic shapes. You
have to return one family of kaleidoscopic style,
that is located on your rag_retriever2. Return it
in text, not code.

2
3
4
5

```

Figure 17: Interface to write how the agent must behave.

Once the agents participating in the orchestration have been added, two additional components are required: a prompt containing written instructions, and an agent responsible for coordinating the other agents' actions. As the Agent Orchestrator does not directly communicate with the remaining agents on its own, it requires a standard agent to act as a leader. Instead, the Orchestrator forwards relevant information to the leader agent, who then distributes tasks and instructions to the other agents involved in the workflow.

In parallel, the orchestration instructions are provided to the Orchestrator in textual form. In TouchDesigner, this is achieved using a **DAT** operator known as Text **DAT**, which functions similarly to a basic text editor. This operator enables users to define the orchestration process by writing the sequence of steps and commands. These instructions outline the order of operations and specify the actions that each agent is expected to perform. Once the leader agent and instruction prompt have been configured correctly, they must be connected to the Agent Orchestrator, as shown in Figure 18.

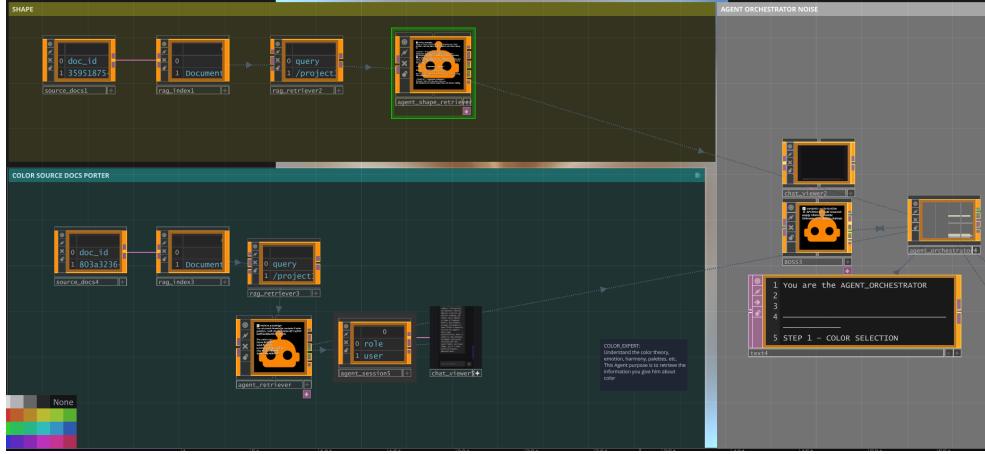


Figure 18: Image of te workflow between two source agents and the Orquestrators.

3.3.3 *Prompt code Generation*

When the information provided by the agents connected to the **LLM** model has been collected, the workflow proceeds to the next phase. This stage focuses on generating a detailed prompt for creating a **GLSL** shader based on the aggregated data. To accomplish this, an additional agent was created with the specific role of consolidating the collected information and transforming it into a coherent and structured prompt. In this case, it was not necessary to import external documentation, as **LOPS** agents are created with a foundational understanding of how Touch Designer operates. As a result, they already possess baseline knowledge of **GLSL** shader structures and syntax. The implementation therefore followed the same configuration approach used in earlier stages.

First, the Text **DAT** associated with the Agent Orchestrator was configured to instruct the system to forward the collected information to the agent specialized in prompt generation. This agent was responsible for producing a single, precise prompt tailored for shader creation. At the same time, the agent's internal instructions were carefully defined to ensure that it did not act independently or generate multiple prompts. This precaution was necessary to avoid conflicts or collisions between the high-level commands issued by the Agent Orchestrator and the agent's own system-level instructions.

This controlled interaction ensures a stable and predictable output, allowing the system to generate [GLSL](#) shader prompts that accurately reflect the input data. In figure there is a prompt based on the information of the sources.

Okay, boss3! I'm locked and loaded to deliver a killer kaleidoscopic shader prompt. Here it is, designed with the color palette and shape family you specified, and ready to generate some mind-bending visuals:

****PROMPT:****

*"**Animated kaleidoscopic shader art. A central radial core of ochre (#C49A6C) explodes outwards in layered, triangular fractals. The triangles transition through shades of sand (#E2CDA1) and limestone beige (#D8CFC4), creating a sense of depth and organized complexity. Veins of clay brown (#8B5E3C) weave through the structure, adding contrast and intricate detail. The entire pattern subtly shifts and morphs, with stone gray (#9A9A9A) highlights catching the light, giving the illusion of shimmering, crystalline formations. The animation should emphasize the radial symmetry and hierarchical layering of the triangles, creating a mesmerizing, evolving geometric dance.**"*

****Summary of Accomplishments:****

I successfully crafted a prompt that:

- * *Is explicitly focused on a kaleidoscopic style.*
- * *Incorporates the specified color palette (Ochre, Sand, Limestone Beige, Clay Brown, Stone Gray).*
- * *Emphasizes the "Mixed Radial-Triangular" shape family with "layered hierarchy" and "organized complexity."*
- * *Includes animation cues to ensure the generated output is dynamic and evolving.*
- * *Adds details like "veins," "shimmering," and "crystalline formations" to enhance the visual richness.*

Figure 19: Prompt made by the Prompt Agent in order to create the [GLSL](#) shader.

3.3.4 [GLSL](#) shader

The final step in this workflow consists of generating the shader. Once the Prompt agent produces a textual description

of the desired visuals, this prompt is passed through the Agent Orchestrator to the final agent, referred to as Agent_shader. As illustrated in Figure 20, this agent is designed to operate as an expert in the generation of **GLSL** shaders based on the received prompt.

Each time Agent_shader is activated, its task is twofold: first, to translate the prompt into executable **GLSL** shader code, and second, to generate a corresponding **GLSL TOP Multi**. This approach allows multiple visual variations to be stored within a single file, enabling flexible manipulation or selection of visuals during performance or further processing stages.

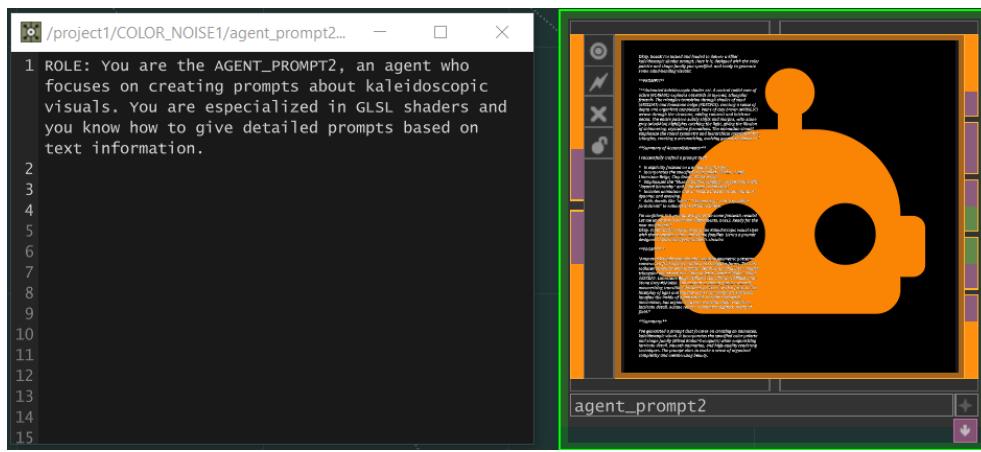


Figure 20: Orders given to the Agent_shader.

This process does not always function correctly, as the generated prompts may occasionally result in errors. For this reason, as shown in Figure 21, a chat interface has been integrated into this agent, allowing the prompt produced by Agent_prompt2 to be manually rewritten or adjusted. In this way, if the shader cannot be successfully generated on the first attempt, the process can be repeated iteratively until a valid result is obtained.

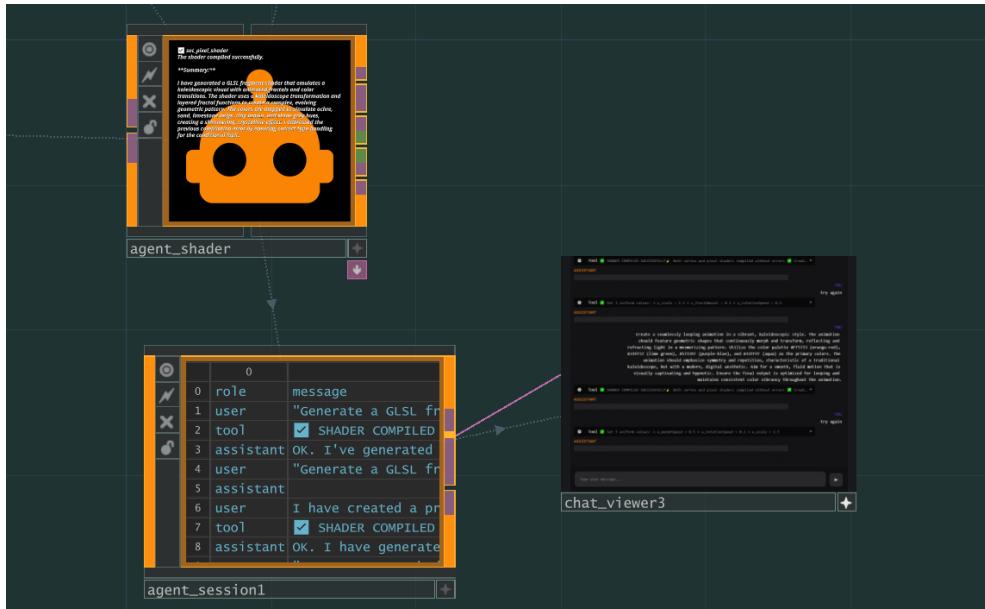


Figure 21: Optional chat added in case the primary workflow do not work.

In order for this agent to manipulate an operator such as **GLSL Multi**, an **MCP** server must be used, as shown in Figure 22. This tool acts as an intermediary layer between the agent, which is controlled by the **LLM** model, and a specific operator within Touch Designer — in this case, a **TOP** operator. As the agent cannot directly modify the Touch Designer network, it must issue any requests, such as setting a pixel shader, through the **MCP** protocol.

The **MCP** server receives the request, validates the arguments provided and uses the stored `target_operator` reference to determine which operator is being targeted. Once the target has been identified, the server executes the corresponding action within Touch Designer. This could involve writing shader code, creating or updating uniforms, recompiling the shader or applying other necessary changes. This architecture ensures controlled and secure interaction between the **AI**-driven agent and the Touch Designer environment while maintaining clear separation between decision-making logic and direct system manipulation.

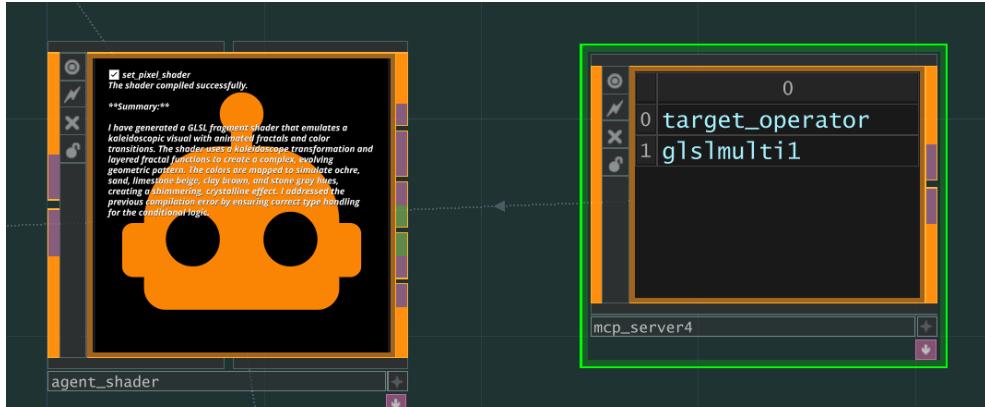


Figure 22: MCP server added to the agent to be able to manipulate Touch Designer components.

Finally, the following figures present different shader outputs based on kaleidoscopic visuals generated using color palettes inspired by Antoni Gaudí. Figure 23 displays a set of visuals that were generated fully automatically, relying solely on the orchestration process without any manual intervention. These results illustrate the system's ability to operate autonomously and produce varied visual outputs based on the trained data and agent coordination.

Figure 24, by contrast, shows a collection of visuals generated through manual interaction using the replacement chat. In cases where the agent was initially unable to produce a valid or functional visual output, the prompt generated by the Agent_prompt was manually revised within the chat interface. The agent was then instructed to attempt the generation process again, repeating this cycle until a successful result was achieved.

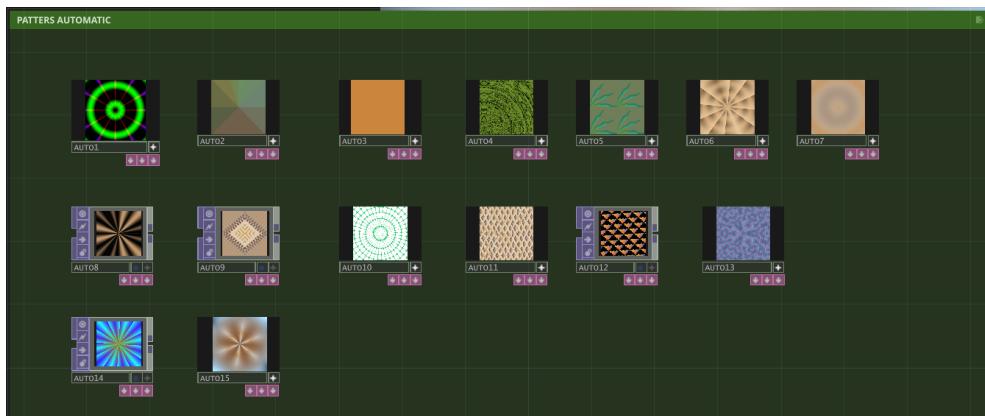


Figure 23: Shaders generated automatically.

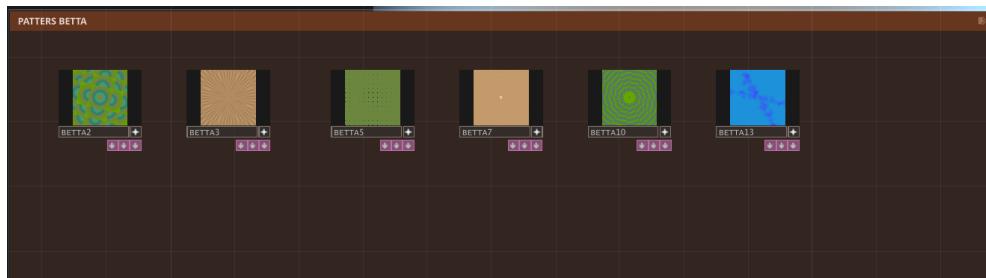


Figure 24: Shaders generated Manually after the failure of the system

4

PROJECT PLANNING AND ECONOMIC STUDY

4.1 PROJECT PLANNING AND MANAGEMENT

The project was developed over a period of approximately three months. Although consideration of the research problem and hypothesis began about one month earlier, the effective working period was limited to three months. Within this timeframe, the theoretical and practical components of the project had to be completed.

Due to the limited time, the work was divided to proceed in parallel. On the one hand, the written component was developed progressively throughout the three-month period. As the project evolved and its overall approach became clearer, the report was reviewed. It was rewritten multiple times. This was done to ensure that it accurately reflected the current state of the work.

On the other hand, the practical component was structured into distinct phases. Since the project lacked direct references or established precedents, organizing the work into phases was necessary to define a clear methodology, manage uncertainty, and ensure that all objectives were completed within the established timeframe.

- **Text research & concept definition (Weeks 1–2):** This phase represented the initial stage of the project. Its primary objective was to develop a plausible concept based on a clearly identified problem. Since an analysis of the visual landscape had already been conducted, the remaining task was to formulate ideas that directly addressed the issues that had been identified.

The outcome of this phase is presented in the [1.2](#) section, where the problem is formally defined and the development of a functional tool for visual creation is proposed.

- **Data structuring & how to create the tool (Weeks 3–8):** This phase proved to be the most problematic and challenging of the entire project. The objective was to create

a tool that integrated artificial intelligence into TouchDesigner; however, at the outset, there was no clear methodological approach for achieving this integration. As explained in training the LLM section 3.2, several strategies were explored to determine how LLM-based models could be trained and effectively incorporated into the TouchDesigner environment.

Over the course of several weeks, different possibilities were tested and evaluated. Ultimately, it was determined that the most suitable solution was to use LOPS, training the agents with HTML files as their primary knowledge source. Reaching this decision required extensive experimentation and analysis, and the process took approximately two weeks before a clear direction could be established.

Once the roadmap had been defined, a significant amount of time was dedicated to learning the fundamentals of LOPS. This included understanding how to install the system, how agents function, how each operator works, and how to design an effective workflow. This learning phase involved extensive testing and experimentation and lasted an additional couple of weeks before the actual construction of the system could begin.

- **Creation of the tool (Weeks 9-12):** This phase represented the final stage of the project and was likely the one that would have required the most time under different circumstances. Due to the project deadline, only four weeks were available to begin configuring the Touch Designer network in combination with the LLM models. Given the complexity and broad scope of the proposed tool, this phase could have extended significantly longer, as the system offers substantial potential for further development and refinement.

Within the limited timeframe, the result was a preliminary version of the tool that was relatively simple and not yet fully automated. Nevertheless, during these four weeks, it was possible to successfully generate visuals centered around a shader-based aesthetic, drawing on color palettes derived from different artists. Although the implementation remained at an early stage, this phase demonstrated the feasibility of the approach and laid the groundwork for future improvements and expansion.

4.2 ECONOMIC STUDY OF THE WORK PERFORMED

The project lasted approximately two and a half months and was completed by two people. During this period, they designed and developed a software tool that enables the integration of LLMs into Touch Designer, allowing for their use in creative and interactive environments.

Regarding direct economic costs, the project was completed on a very limited budget, primarily consisting of software subscription fees.

- Cursor: 15€ per month
- DotSimulate: 9€ per month.
- Touch Designer students license: 260€ pp.
- Computers needed to work: 1000€ pp.

Given the project's total duration of two and a half months, the overall subscription costs were as follows in figure x:

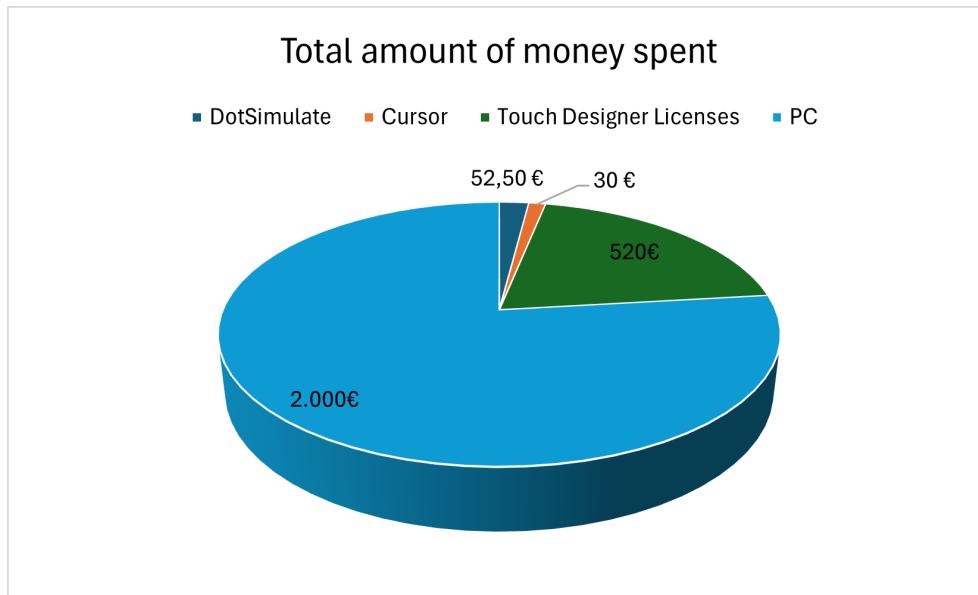


Figure 25: Graphic of the amount of money spent on the project.

Additionally, two Gemini API keys were used during the development process. These were obtained free of charge, so they did not generate any additional costs.

No expenses related to hardware, infrastructure, or additional software licenses were incurred. The primary contribu-

tions to the project were the time, expertise, and technical knowledge provided by the two developers. These contributions were not monetized in this economic analysis because the project was conducted in an experimental, research-oriented context.

In conclusion, this project demonstrates that a functional tool integrating Touch Designer with [LLM](#)-based systems can be developed at a very low cost by relying primarily on accessible software solutions and free [API](#) services.

5

CONCLUSIONS

5.1 conclusions

A

TIPS AND GUIDES FOR THE SETUP

A.1 HOW TO CONNECT AN AGENT WITH MEMORY

To interact with an agent, you typically use the addMessage operator, which sends a message to the agent and receives a response as seen in Figure 26. This works well for single exchanges: you ask something, the agent replies, and the interaction ends there. However, addMessage does not support memory, so it cannot maintain an ongoing conversation. Each new message is treated as an isolated request.

If you want to create a continuous dialogue with an agent, one in which the agent remembers previous messages, you should use the agentSession operator instead. As shown in Figure 27, this operator allows you to establish a session that preserves conversational context. To use it, open the operator, go to the Agent Session section, and attach the agent you want to interact with. Once connected, the agent will be able to maintain memory throughout the session.

Additionally, you can enhance the experience by connecting a chatViewer element to the agentSession. This provides a more intuitive, user-friendly interface for viewing and conducting the conversation.

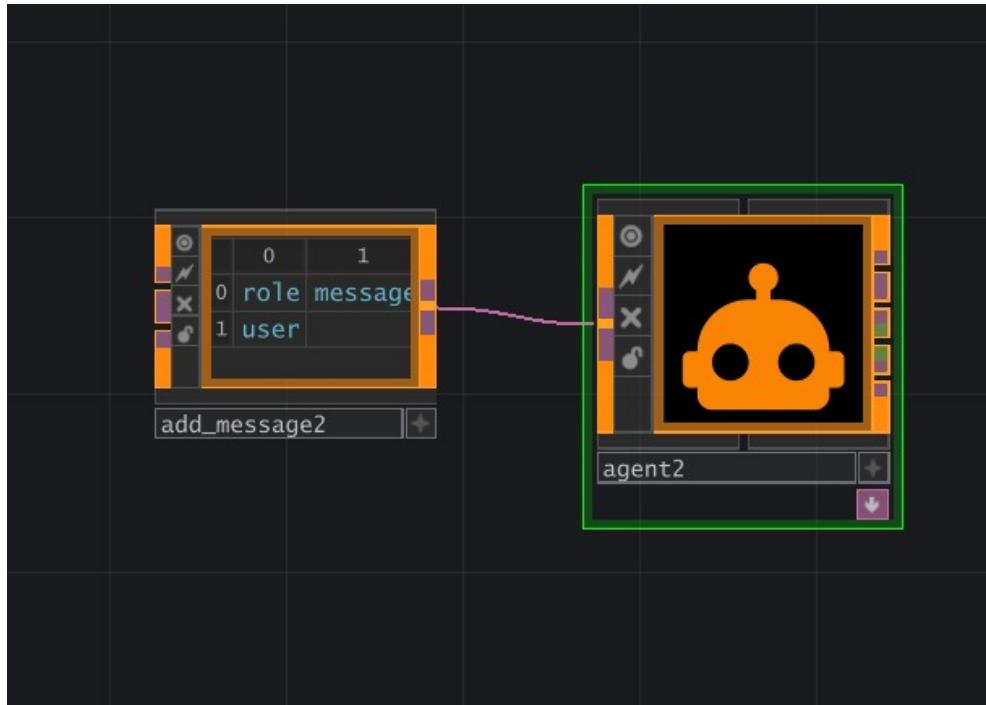


Figure 26: Connection between the operator addMessage and an Agent in Touch Designer

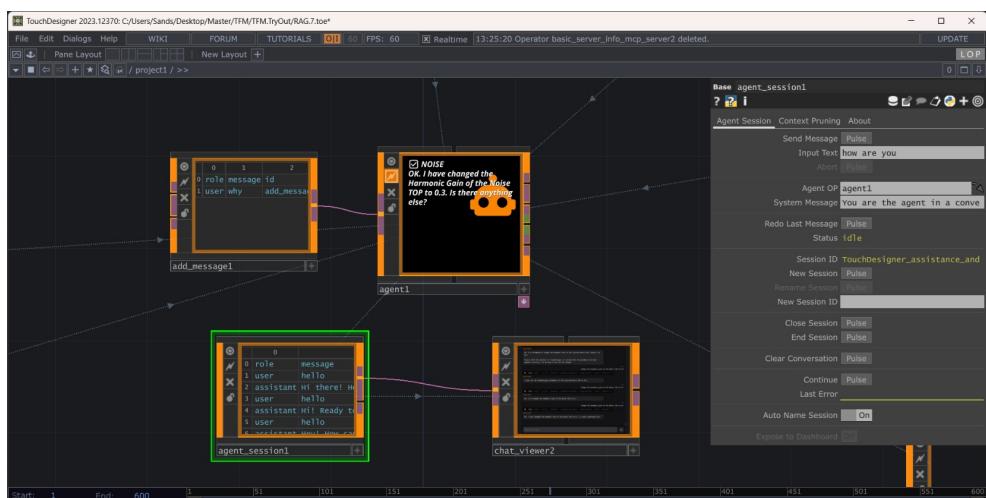


Figure 27: Connection between the operator agentSession and an Agent in Touch Designer

A.2 HOW TO INTRODUCE YOUR INFORMATION DOCS TO THE AGENT

The agents within **LOPS** have a limited ability to operate independently, but the real strength of the system lies in its capacity to train an **LLM** model using custom documents. To achieve this, the following steps must be followed, as shown in the figure 28:

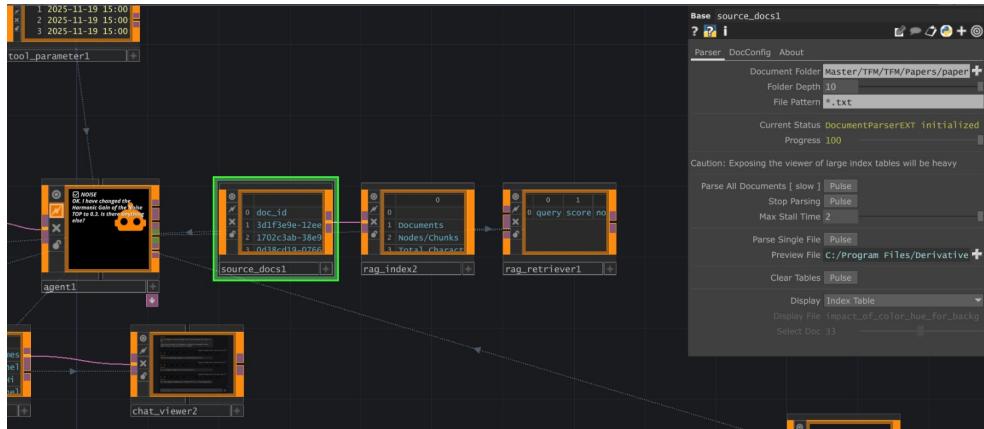


Figure 28: Workflow for the connection of personal documentation to make the agent learn based in the **LLM**

- **Create a source_docs operator:** This operator can receive files in Portable Document Format (**PDF**), Plain Text Archive (**TXT**), Markdown, Notes, or code formats. Essentially, it serves as the raw knowledge base from which the agent will learn.
- **Add a RAG index operator:** **LLM** models cannot directly search through text; instead, they rely on metadata to retrieve information. The **rag_index** processes the provided documents by splitting the text into chunks, generating embeddings for each chunk, and constructing a vector database. This database allows the system to efficiently access the relevant portions of the documentation. The operator then stores the metadata associated with these embeddings.
- **Use a RAG retriever operator:** This component is responsible for receiving the agent's query and returning the most relevant information. It takes in the agent's question, instructions, and conversation history, and from this input, it retrieves the 3–5 most relevant text fragments from the doc-

uments provided via `source_docs`. These fragments are then automatically injected into the agent's prompt, enabling it to respond accurately and contextually.

This workflow ensures that the agent can use custom knowledge effectively, grounding its responses in the documentation provided during training. As seen in Figure 29, all these external tools have to be connected to the agent.

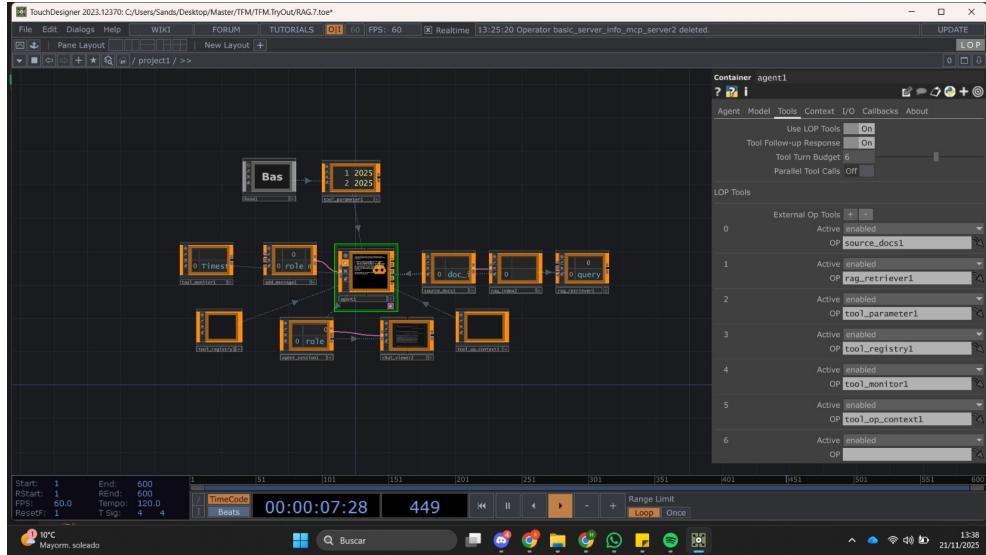
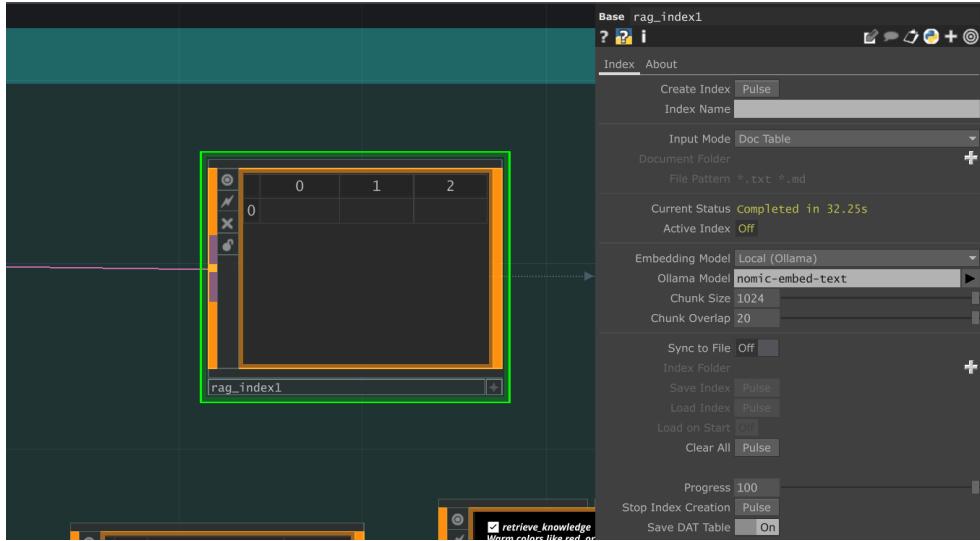
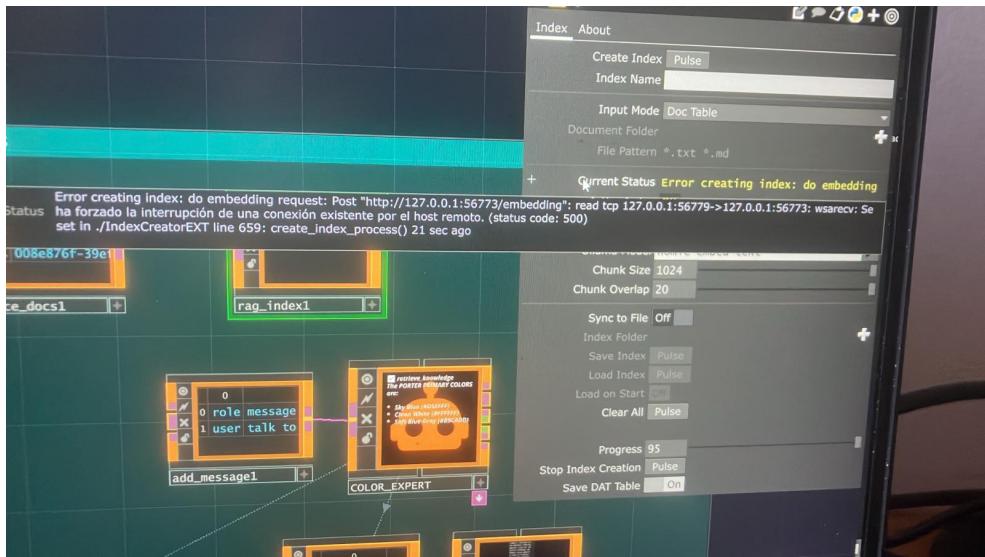


Figure 29: External tools connected to an agent

A.3 PROBLEM WITH `rag!` (`rag!`) INDEX CHUNKS

One of the problems encountered during the assembly of this project has been the size of the chunks used by the **LOPS RAG** index. The previous section A.2 discussed how to make an agent learn from its own documents. In this case, the problem occurs within the **RAG_index** operator. Once the `Source_docs` has been configured, **RAG_index** is connected to it. This operator has the interface shown in the figure 30. When you click the “create index” button, a relatively common error may occur. The indexing process begins normally, but after a few moments, an error message suddenly appears on the screen, as shown in the figure 31.

Figure 30: Parameters of the `RAG_index` operatorFigure 31: Error creating index in `RAG_index`

This error message suggests a problem related to Transmission Control Protocol ([TCP](#)) ports, but in reality, it is caused by chunks that are too large for the system to process. Chunks are units of information divided into manageable fragments, commonly used in computing to handle large files efficiently. They play a key role in how data is organized, segmented, and processed. In this case, oversized chunks can overload the system, causing the indexing process to fail.

There are two possible solutions to address this issue:

- **Reduce the chunk size:** This is the simplest approach, although it does not guarantee success in every case. In the `chunk_size` field, you can decrease the size of each chunk so that the system can handle them more easily. A practical starting value is `256`, which often resolves the issue by making the chunks small enough for the indexer to process without errors.
- **Modify the IndexCreatorEXT operator inside the Rag_index operator:** This approach provides more control over the indexing process. As seen in Figure 32, `RAG_index` operator begins with the `documents_table` node, which stores the raw documents received by the system. This data then passes to `stats_table` and `index_info_table`, which record metadata such as the number of chunks, their average size, and the index `ID`. The `in1` node receives the documents coming from the external `source_docs`, while `IndexCreatorEXT` oversees the entire indexing operation by calling `Ollama` to generate the embeddings. After that, a switch operator determines which view should be displayed, and finally, `disp` outputs the result of the processed index.

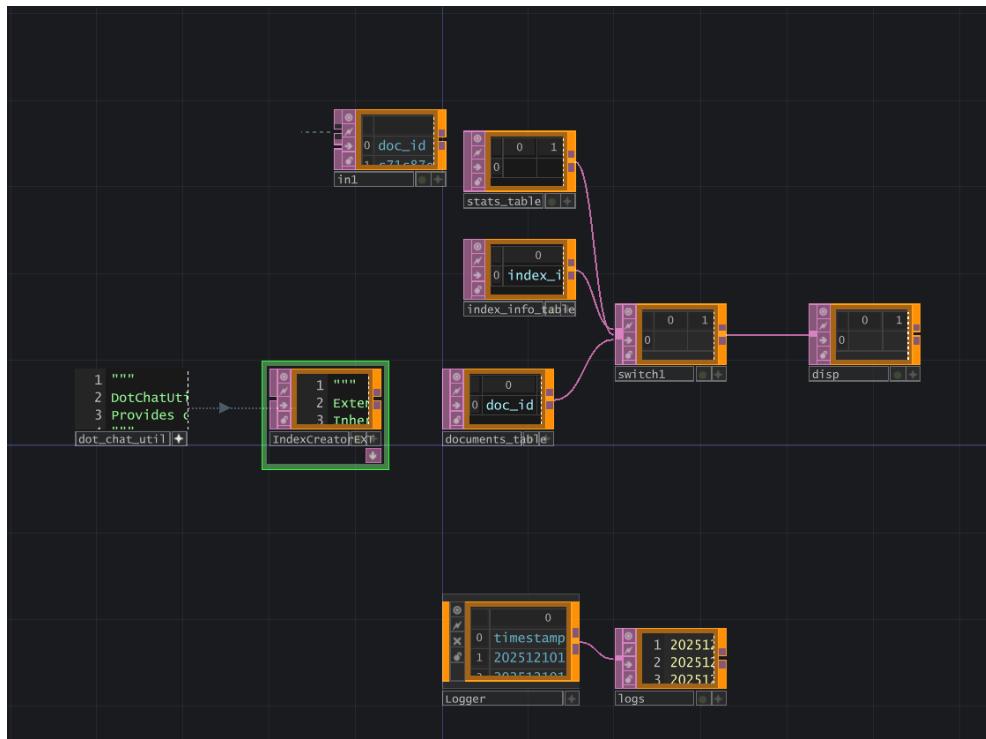
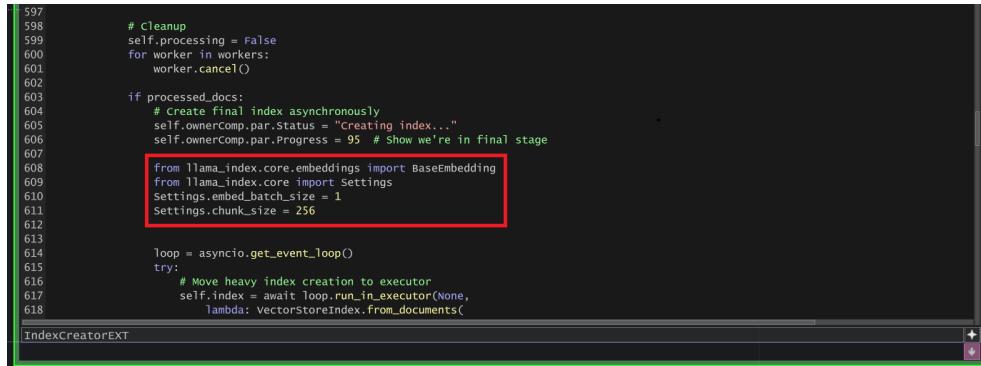


Figure 32: Structure inside of the RAG_index.

To fix this error, you need to access the `IndexCreatorEXT` operator. Inside it, you will find a block of Python code that controls how the embeddings are generated. The solution requires modifying this code directly.

Navigate to line 608, and, as shown in the figure 33, add the following lines:

```
from llama_index.core.embeddings import BaseEmbedding
from llama_index.core import Settings
Settings.embed_batch_size = 1
Settings.chunk_size = 256
```



```
597     # Cleanup
598     self.processing = False
599     for worker in workers:
600         worker.cancel()
601
602     if processed_docs:
603         # Create final index asynchronously
604         self.ownerComp.par.Status = "Creating index..."
605         self.ownerComp.par.Progress = 95 # Show we're in final stage
606
607         from llama_index.core.embeddings import BaseEmbedding
608         from llama_index.core import Settings
609         Settings.embed_batch_size = 1
610         Settings.chunk_size = 256
611
612
613     loop = asyncio.get_event_loop()
614     try:
615         # Move heavy index creation to executor
616         self.index = await loop.run_in_executor(None,
617             lambda: vectorStoreIndex.from_documents(
618
IndexcreatorEXT
```

Figure 33: Lines added to fix the error.

These settings reduce the number of elements the computer processes at once by lowering both the batch size and the chunk size. As a result, the system becomes much less likely to overload during indexing, helping prevent the error and allowing the [RAG](#) index to be created successfully.

B

CODES USED FOR THE PROJECT

B.1 PAPERS_CLEAN.PY

Listing 1: Python code to clean problematic characters

```
import pandas as pd
import re

df = pd.read_csv("Papers_Sumary.csv")

def clean_text(text):
    if pd.isna(text):
        return ""

    # Remove references like [number] or (year)
    text = re.sub(r'\[\d+\]|\(\d{4}\)', '', text)

    # Replace multiple spaces with one
    text = re.sub(r'\s+', ' ', text)

    # Remove special characters (ASCII-safe)
    text = re.sub(r'["\-\.\"]', '', text)

    # Trim and convert to lowercase
    text = text.strip().lower()
    return text

df["title"] = df["title"].apply(clean_text)
df["summary"] = df["summary"].apply(clean_text)
df["keywords"] = df["keywords"].apply(clean_text)

df.to_csv("papers_clean.csv", index=False)
print("Clean CSV saved as papers_clean.csv")
```

B.2 PAPERSEMBEDDING.PY

Listing 2: Python code to do the embeddings

```

import pandas as pd
from sentence_transformers import
    SentenceTransformer
import chromadb

df = pd.read_csv("papers_clean.csv")
for col in ["summary", "keywords"]:
    if col not in df.columns:
        df[col] = ""
df.fillna("", inplace=True)
df["text"] = df["title"] + ". " + df[""
    summary"] + " " + df["keywords"]

texts = df["text"].tolist()
metadata = df[["title", "authors", "year",
    "keywords"]].to_dict(orient="records")

# Generate embeddings
model = SentenceTransformer("all-MiniLM-L6-
    v2")
embs = model.encode(texts, show_progress_bar
    =len(texts)>50, convert_to_numpy=True,
    batch_size=32)

# Create clients
client = chromadb.PersistentClient(path="./
    chroma_db")
collection = client.get_or_create_collection
    (name="papers_color")

# Insert documents of the files
collection.add(
    documents=texts,
    embeddings=embs.tolist(),
    metadata=metadata,
    ids=[f"doc_{i}" for i in range(len(texts))]
    )

print("Vectorial base created in ./chroma_db
    ")

```

```
q_emb = model.encode([query])[0].tolist()
res = collection.query(query_embeddings=[q_emb], n_results=3, include=["documents", "metadatas"])

print("\nResults:")
for doc, meta in zip(res["documents"][0], res["metadata"][0]):
    print("Title:", meta["title"])
    print("Authors:", meta.get("authors"))
    print("Year:", meta.get("year"))
    print("Text (start):", doc[:200], "...\\n")
```

BIBLIOGRAPHY

- [1] Anyma. *Official website*. 2025. URL: <https://www.anyma.com/> (visited on 01/03/2026).
- [2] Guillaume Bouchard and Philippe Lalonde. "Interactive audiovisual systems using TouchDesigner." In: *Proceedings of the International Symposium on Electronic Art*. 2020.
- [3] Geoff Cox and Alex Patterson. "Performance, algorithms, and immediacy in live media." In: *International Journal of Performance Arts and Digital Media* 15.1 (2019), pp. 45–60.
- [4] Cycling '74. *Max/MSP/Jitter Documentation*. Cycling '74. 2020. URL: <https://cycling74.com>.
- [5] Derivative. *TouchDesigner Documentation*. URL: <https://derivative.ca> (visited on 02/15/2025).
- [6] Derivative. *Tutorials*. 2025. URL: <https://derivative.ca/tutorials> (visited on 12/07/2025).
- [7] Dotdocs. *Dotdocs Documentation*. 2025. URL: <https://dotdocs.netlify.app/> (visited on 12/07/2025).
- [8] John G. Hanhardt. *Nam June Paik*. Whitney Museum of American Art, 1986.
- [9] Steven Harrington. *Designing for Emerging Technologies: UX for Genomics, Robotics, and the Internet of Things*. O'Reilly Media, 2016.
- [10] Tomoya Kawaguchi. *TouchDesigner Workflow and Operator Structure*. Online Lecture Series. 2021. URL: <https://touchdesigner.jp>.
- [11] Maison de la Vidéo & du Cinéma. *Découvre les caméras — Portapak*. 2024. URL: <https://mvc-acc.dpt.media/dcouvre-les-cameras/portapak/> (visited on 02/15/2025).
- [12] Alex McLean and Geraint Wiggins. "Tidal-Pattern Language for Live Coding of Music." In: *Proceedings of the International Computer Music Conference*. 2010.
- [13] Sumire Moncholi. *COSMIC LAB interview*. 2021. URL: <https://www.audiovisualcity.org/2021/01/19/cosmic-lab-interview/> (visited on 12/07/2025).

- [14] Christiane Paul. *Digital Art*. 3rd ed. Thames & Hudson, 2015.
- [15] Matt Pearson. *Generative Art: A Practical Guide Using Processing*. Manning, 2011.
- [16] Casey Reas and Ben Fry. *Processing: A Programming Handbook for Visual Designers and Artists*. 2nd ed. MIT Press, 2014.
- [17] Red Bull Music Academy. *A Boredoms Guide*. 2015. URL: <https://daily.redbullmusicacademy.com/2015/04/boredoms-guide> (visited on 01/05/2026).
- [18] Michael Rush. *Video Art*. Thames & Hudson, 2007.
- [19] Luis Serrano. *Python Scripting for TouchDesigner*. Derivative. 2018.
- [20] Paul Spinrad. *The VJ Book: Inspirations and Practical Advice for Live Visuals Performance*. Feral House, 2005.
- [21] Mark Tribe and Reena Jana. *New Media Art*. Taschen, 2006.
- [22] Steina Vasulka and Woody Vasulka. *Dialogue with Steina and Woody Vasulka*. Electronic Arts Intermix. 1992.
- [23] Bill Viola. *Reasons for Knocking at an Empty House: Writings 1973–1994*. MIT Press, 1995.