# Dalledress – Better Blockies Through Science

## Introduction

In this paper, we describe an algorithm that transforms hexadecimal input strings into complex generative artworks, subsequently minted as non-fungible tokens (NFTs) on the Ethereum blockchain.

The algorithm segments its input to derive a collection of attributes from predefined, version-controlled databases. These attributes are then assembled into a structured prompt, which is enhanced through an AI-driven literary stylization before being visualized as a unique artwork. The NFT allows for the cryptographic minting of the artwork, ensuring that it is embedded with immutable provenance and first-owner data.

The synthesis of a deterministic data pipeline with creative AI image generation and blockchain technology pioneers not only a new method of digital art creation but also introduces a scalable, secure mechanism that may enhance digital authentication through the production of unique, easily identified “identicons” or “blockies”.

## An Example

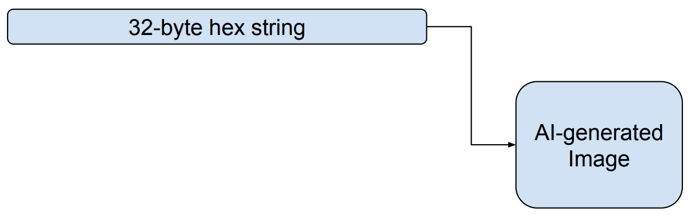
To make the idea more obvious, we present an example of one such generated image:

|  |  |  |
| --- | --- | --- |
|  |  |  |

## Format of the Paper

This paper is organized into sections that align with the various stages of the algorithm. The algorithm advances through distinct steps: the *Chopper*, the *Selector*, the *Prompt Builder*, the *Prompt Enhancer*, and, finally, the *Image Generator*. The output of one process feeds into the input of the next in a data pipeline, allowing for a high degree of concurrency.

At the highest level, the pipeline accepts a hexadecimal string (for example, an Ethereum address, a block hash, or a transaction hash) and outputs an AI-generated image.



Looking more deeply, one sees that the process is carried out in the following order:

A diagram of a structure

Description automatically generated

The paper is organized along these lines. It will become apparent as we proceed why we present the processes on two lines. In short, we distinguish between those processes that are deterministic and those that are not. This has implications for the use of the output as a reproducible identicon.

We begin by describing the process of chopping.

## The Chopper

The first process, which is deterministic, is called the *Chopper*. The *Chopper* accepts a hexadecimal character string as input and returns a collection of fifteen (15) substrings represented as three-byte, fixed-width character strings.

func Chopper(input string) [15][3]string {

...

}

By interface, the *Chopper* accepts any string, but the current system requires either an ENS name, a 20-byte hexadecimal Ethereum address, or a 32-byte hexadecimal hash (such as a transaction hash). ENS names (strings ending with .eth) are converted to 20-byte addresses if possible (or the empty string if not possible). Strings shorter than 20-bytes are rejected. Strings longer than 32-bytes are truncated to 32 bytes (64 characters).

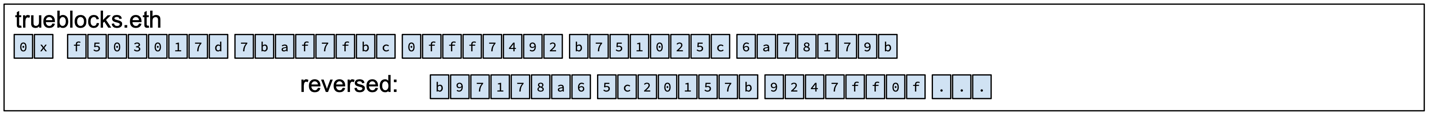
If an input string is (or resolves to) 20-bytes long, the string is extended to 32-bytes in the following manner. First, a copy of the string is created and reversed. Next, the reversed string is appended to the original string resulting in a 40-byte string. Finally, the 40-byte string is truncated to 32-bytes. In this way, all input, after being converted, becomes a fixed-width 32-byte string, and the *Chopper* may proceed.

Given a fixed-width, 32-byte string, the *Chopper* segments the string into fifteen three-byte substrings, These substrings are sent to the following process, *The Selector*, which is detailed below. The start byte for each substring is determined by the following table. All substrings are three bytes long. The names assigned to each substring will become apparent in a moment. Note that the substrings overlap.

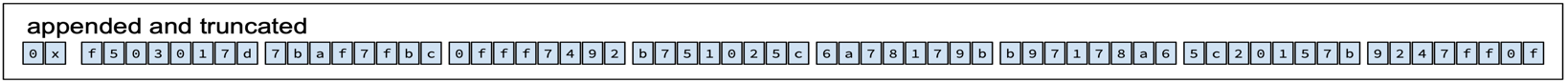
|  |  |  |  |
| --- | --- | --- | --- |
| substring | start | nBytes | attribute |
|  | 0 | 3 | adverb |
|  | 2 | 3 | adjective |
|  | 4 | 3 | noun |
|  | 6 | 3 | emotion |
|  | 8 | 3 | occupation |
|  | 10 | 3 | action |
|  | 12 | 3 | artistic style 1 |
|  | 14 | 3 | artistic style 2 |
|  | 16 | 3 | literary style |
|  | 18 | 3 | color 1 |
|  | 20 | 3 | color 2 |
|  | 22 | 3 | color 3 |
|  | 24 | 3 | orientation |
|  | 26 | 3 | gaze |
|  | 28 | 3 | background style |

### An example

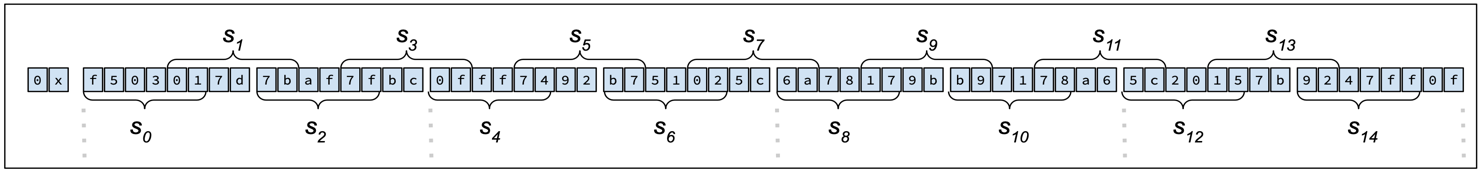
As an example, suppose the input string is trueblocks.eth, an ENS name associated with the address 0xf503017d7baf7fbc0fff7492b751025c6a78179b. The Chopper first converts the ENS name into an address and then makes a copy and reverses it:



It then appends the reversed copy to the end of the original string and truncates the result to 32-bytes. (Note that an already 32-byte transaction hash may skip these initial steps.)



From here, the *Chopper* completes its work by segregating the bytes into the fifteen substrings as follows.



The final output is this collection of hexadecimal values:

|  |  |  |
| --- | --- | --- |
| substring | attribute | bytes |
|  | adverb | 0xf50301 |
|  | adjective | 0x017d7b |
|  | noun | 0x7baf7f |
|  | emotion | 0x7fbc0f |
|  | occupation | 0x0fff74 |
|  | action | 0x7492b7 |
|  | artistic style 1 | 0xb75102 |
|  | artistic style 2 | 0x025c6a |
|  | literary style | 0x6a7817 |
|  | color 1 | 0x179bb9 |
|  | color 2 | 0xb97187 |
|  | color 3 | 0x87a6c5 |
|  | orientation | 0xc50215 |
|  | gaze | 0x157b29 |
|  | background style | 0x2947ff |

## A Note About Attributes

We mention above certain named attributes. Before proceeding we wish to describe these attributes.

The system includes a database, , consisting of twelve (12) tables, , called the *Attribute Tables*. The *Attribute Tables*, which correspond to the above substrings, contain words or phrases of a particular part of speech (such as *nouns*, *adjectives*, or *adverbs*), the all-important *emotional* table containing various human emotions, action-oriented attributes (such as *occupation* and *action*), stylistic components (such as *artistic*, *literary*, *background*, or *color*), or other collections of attributes including *orientation* and *gaze* which are explained more fully below.

These database tables are carefully version controlled and published globally to IPFS; a ubiquitous content-addressable file store available (permissionlessly) to anyone who wishes to access it through the IPFS daemon. More information about these databases, along with information on how to obtain the tables (and share them), are available in the Appendix.

## The Selector

We now proceed to describe the second deterministic step in the pipeline, the *Selector*.

The *Selector* accepts as input fifteen three-byte substrings from the *Chopper* and outputs a collection of fifteen attributes chosen from the corresponding *Attribute Tables*.

func Selector([15][3]byte) [15]string {

...

}

The *Selector* converts each substring, , into its equivalent representation as an unsigned integer, , through a simple hexadecimal conversion. Dividing by scales the integer into a floating-point factor between 0.0 and 1.0. Multiplying by the number of records in the appropriate table,, and taking the floor ensures that the selection is evenly distributed across the database records in the table. Note that multiple values are selected from the artistic style and color databases.

Pseudo-mathematically, the *Selector* process may be written as:

This process is detailed in the following example:

| substring | attribute | bytes |  |  |  | selector |
| --- | --- | --- | --- | --- | --- | --- |
|  | adverb | 0xf50301 | 16,057,089 | 0.9571 | 3,052 | 2,920 |
|  | adjective | 0x017d7b | 97,659 | 0.0058 | 1,295 | 7 |
|  | noun | 0x7baf7f | 8,105,855 | 0.4831 | 3,470 | 1,676 |
|  | emotion | 0x7fbc0f | 8,371,215 | 0.4990 | 279 | 139 |
|  | occupation | 0x0fff74 | 1,048,436 | 0.0625 | 220 | 13 |
|  | action | 0x7492b7 | 7,639,735 | 0.4554 | 1,035 | 471 |
|  | artistic style 1 | 0xb75102 | 12,013,826 | 0.7161 | 325 | 232 |
|  | artistic style 2 | 0x025c6a | 154,730 | 0.0092 | 325 | 2 |
|  | literary style | 0x6a7817 | 6,977,559 | 0.4159 | 95 | 39 |
|  | color 1 | 0x179bb9 | 1,547,193 | 0.0922 | 139 | 12 |
|  | color 2 | 0xb97187 | 12,153,223 | 0.7244 | 139 | 100 |
|  | color 3 | 0x87a6c5 | 8,890,053 | 0.5299 | 139 | 73 |
|  | orientation | 0xc52015 | 12,918,805 | 0.7700 | 8 | 6 |
|  | gaze | 0x157b29 | 1,407,785 | 0.0839 | 8 | 0 |
|  | background style | 0x2947ff | 2,705,407 | 0.1613 | 8 | 1 |

The final step of the process is to select the attribute values from the tables. Note that in some cases, the selected attribute contains more than one field. For example, the nouns database (which happen to all be animals) consists of records with the follow fields.

common name, class, family

This allows for greater expressivity in the output without over-extending the number of attributes. See the JSON data produced in the *Structure Builder* section below. We conclude this section by furthering the example.

### Example

| substring | attribute | selector | attribute value |
| --- | --- | --- | --- |
|  | adverb | 2,920 | |  |  | | --- | --- | | adverb: | wimpishly | | definition: | in a manner that is weak or cowardly | |
|  | adjective | 7 | |  |  | | --- | --- | | adjective: | accomplished | | definition: | highly skilled or proficient | |
|  | noun | 1,676 | |  |  | | --- | --- | | animal: | indochinese tiger | | order: | carnivora | | family: | felidae | |
|  | emotion | 139 | |  |  | | --- | --- | | emotion: | ilinx | | group: | excitement | | polarity: | positive | | language: | English | | description: | the strange excitement of wanton destruction; a sensation of spinning falling and losing control | |
|  | occupation | 13 | |  |  | | --- | --- | | occupation: | beautician | | description: | beauticians provide cosmetic treatments and services to enhance appearance. | |
|  | action | 471 | |  |  | | --- | --- | | action: | knowing | | definition: | having knowledge | |
|  | artistic style 1 | 232 | |  |  | | --- | --- | | style: | postwar shin-hanga | | group: | contemporary and emerging styles | | description: | postwar shin-hanga revives traditional Japanese woodblock printing with modern themes | |
|  | artistic style 2 | 2 | |  |  | | --- | --- | | style: | abstract expressionism | | group: | modern western art movements | | description: | abstract expressionism emphasizes spontaneous automatic or subconscious creation | |
|  | literary style | 39 | |  |  | | --- | --- | | style: | fiction | | description: | creative narratives that are imagined and not based on real events | |
|  | color 1 | 12 | |  |  | | --- | --- | | name: | cadetblue | | hexValue: | #5f9ea0 | |
|  | color 2 | 100 | |  |  | | --- | --- | | name: | palegoldenrod | | hexValue: | #eee8aa | |
|  | color 3 | 73 | |  |  | | --- | --- | | name: | lightyellow | | hexValue: | #ffffe0 | |
|  | orientation | 6 | |  |  | | --- | --- | | orientation: | diagonally | | symmetry: | symmetrically | |
|  | gaze | 0 | |  |  | | --- | --- | | gaze: | into the camera | |
|  | background style | 1 | |  |  | | --- | --- | | style: | The background should be this color {{.Color3.Val}} and pay homage to this style {{.ArtStyle2.Val}} | |

The fifteen part output of the *Selector* process is passed on to the next step in the process, the *Structure Builder*.

## The Structure Builder

This trivial, deterministic step accepts the fifteen attributes produced by *The Selector* and returns a Go struct, called a Dalledress, collating this information.

func StructureBuilder([15]string) Dalledress {

...

}

We present the structure in the following JSON object. This structure is used in subsequent processing.

{

"adverb": "wickedly",

"adjective": "abundant",

"noun:": {

"animal": "indigo snake",

"order": "reptilia",

"family": "colubridae"

},

"emotion": {

"emotion": "ikigai",

"group": "contentment",

"polarity": "positive",

"language": "Japanese",

"description": "the feeling that life is ‘good and meaningful’ and that it is ‘worthwhile to continue living’; reason for being"

},

"occupation": {

"occupation": "audiologist",

"description": "audiologists diagnose and treat hearing disorders"

},

"action": "knocking",

"artisticStyle1": {

"style": "post-war realism",

"group": "contemporary and emerging styles",

"description": "post-war realism depicts everyday life with a focus on social and political themes after World War II"

},

"artisticStyle2": {

"style": "aboriginal art",

"group": "regional and folk art traditions",

"description": "aboriginal art encompasses traditional and contemporary works by indigenous peoples"

},

"literaryStyle": {

"style": "feminist literature",

"description": "explores themes of gender equality and women's experiences"

},

"color1": {

"name": "brown",

"hexValue": "#a52a2a"

},

"color2": {

"name": "orangered",

"hexValue": "#ff4500"

},

"color3": {

"name": "lightcyan",

"hexValue": "#e0ffff"

},

"orientation": "horizontal",

"gaze": "direct",

"background": "lightcyan"

}

It is important to note that each of the proceeding processes and the following process are deterministic. That is, given the same input, each of these processes produces the same output. (As long as the database tables remain unchanged—which is why we store the databases in IPFS—we want to make sure you use the same databases as we’ve used.) This is important if one wishes to use parts of this system for identicons.

We now proceed to describe the final deterministic step in the process: *The Prompt Builder*.

## The Prompt Builder

*The Prompt Builder* uses the templating capability in the Go language to build a prompt that is first enhanced by the AI based on a literary style and then passed back to the AI for the final image generation. Prior to that, we use the following template to render the deterministically produced text for the basic prompt.

var prompt = `Draw a human-like and

{{.Adverb}} {{.Adjective}} {{.Noun}} feeling {{.EmotionShort}}{{.Ens}}.

Noun: human-like {{.Noun}}.

Emotion: {{.Emotion}}.

Primary style: {{.ArtisticStyle}}.

Use only the colors {{.Color1}} and {{.Color2}}.

{{.Orientation}}.

{{.Background}}.

Expand upon the most relevant connotative meanings of  
{{.Noun}}, {{.Emotion}}, {{.Adjective}}, and

{{.Adverb}}. Find the visual representation that most

closely matches the description. Focus on the Noun, the Emotion,

and Primary style.

{{.LiteraryStyle}}

DO NOT PUT ANY TEXT IN THE IMAGE.`

In the above, most replacements take on their obvious values. A few values need clarification. The tag for .EmotionsShort is replaced with only the value field of the Emotions object. The .Emotions tag is replaced with a space-separated join of the value and description fields.

The .Ens tag is replace with the original input if and only if the original input ends with .eth.

The text DO NOT PUT ANY TEXT IN THE IMAGE is an attempt to get the AI to refrain from generating text in the image, something it tends to do unbidden.

The .Orientation and .Background require further explanation. Both tags are first converted into a number between 0 and 7.

.Orientation dictates the direction of the “through-line” in the image and the “gaze” of the main protagonist (that is, the .Noun) based on the values from the following table.

|  |  |  |
| --- | --- | --- |
| orientation | through line | gaze |
| 0 | right | left |
| 1 | upper-right | lower-left |
| 2 | top | bottom |
| 3 | upper-left | lower-right |
| 4 | left | right |
| 5 | lower-left | upper-right |
| 6 | bottom | top |
| 7 | lower-right | upper-left |

The “through-line” might be, for example, the major gesture being carried out by the .Noun. The gaze refers to the direction in which the .Noun is looking. A separate template is used to generate the .Orientation string.

var orientation = `The primary action of the image should be

towards the {{.Orientation.ThroughLine}} of the image.

The {{.Noun}}’s gaze should be in the direction of

the {{.Orientation.Gaze}} of the image.

Note that due to the nature of the AI, it determines exactly what these .Orientation values mean. This is one of the aspects that make the last two processes in whole system (the *Prompt Enhancer* and the *Image Generator*) non-deterministic.

The .Background tag is produced similarly to .Orientation using a table of values:

|  |  |
| --- | --- |
| background | value |
| 0 | transparent |
| 1 | upper-right |
| 2 | top |
| 3 | upper-left |
| 4 | left |
| 5 | lower-left |
| 6 | bottom |
| 7 | lower-right |

switch dd.Background.Num {

case 0:

dd.Background.Val = "The background should be transparent"

case 1:

dd.Background.Val = "The background should be this color {{.Color3.Val}} and pay homage to this style {{.ArtStyle2.Val}}"

case 2:

dd.Background.Val = " The background should be this color {{.Color3.Val}} and subtly patterned"

case 3:

fallthrough

case 4:

fallthrough

case 5:

fallthrough

case 6:

fallthrough

case 7:

dd.Background.Val = "The background should be solid and colored with this color: {{.Color3.Val}}"

default:

to the extent the AI is able to render it. For example, a value of 1 dictates to the AI to make the trough-line of the image to the upper right. This table dictates the .Orientation. We allow the AI to determine exactly what through-line means.

|  |  |  |
| --- | --- | --- |
| orientation | through line | gaze |
| 0 | right | left |
| 1 | upper-right | lower-left |
| 2 | top | bottom |
| 3 | upper-left | lower-right |
| 4 | left | right |
| 5 | lower-left | upper-right |
| 6 | bottom | top |
| 7 | lower-right | upper-left |

## Pseudo-Scientific Mathematical Notation

For each segment *Sj* of the hexadecimal string, mapped to a database *Di*:

*Pi* = *Di*[(int(*Sj*, 16) mod *Ni*)]

where:

*Pi* represents the selected record for the *i*-th category in the prompt.

*Di* is the *i*-th database corresponding to one of the categories (e.g., adverb, adjective, etc.).

*Sj* is the *j*-th segment of the hexadecimal string int(*Sj*, 16) converts segment *Sj* from hexadecimal to decimal.

*Ni* is the number of records in *Di*

\mod denotes the modulo operation, ensuring the selection index is within the bounds of *Di*'s size.

The final prompt structure is then given by concatenating all *Pi*:

Prompt = *P*1 ∣∣ *P*2 ∣∣…∣∣ *P*9

where ∣∣ denotes concatenation of the selected records to form the complete prompt.

## Further Notes on the Attribute Tables

Discuss each database table separately.

Talk about the fields in each

Some fields are used in EmotionsShort EmotionsLong way

Others are used as filters

The final step in this simple algorithm concatenates the selected records into a structured prompt, where each field represents an element as defined by its associated category in the input sequence. This method not only allows for a deterministic but diverse generation of prompts based on the input hexadecimal string. This also ensures that the variation in the number of records across databases is accounted for, making the algorithm adaptable to databases of different sizes. The process exemplifies a creative intersection between mathematical operations and database querying, leveraging the inherent properties of hexadecimal strings and modular arithmetic to dynamically generate structured data representations.

Template Prompt

var promptTemplate = `Draw a human-like and {{.Adverb.Val}} {{.Adjective.Val}} {{.Noun.Val}} feeling {{.EmotionShort.Val}}{{.Ens}}.

Noun: human-like {{.Noun.Val}}.

Emotion: {{.Emotion.Val}}.

Primary style: {{.Style.Val}}.

Use only the colors {{.Color1.Val}} and {{.Color2.Val}}.

{{.Orientation.Val}}.

{{.Background.Val}}.

Expand upon the most relevant connotative meanings of {{.Noun.Val}}, {{.Emotion.Val}}, {{.Adjective.Val}}, and {{.Adverb.Val}}.

Find the representation that most closely matches the description.

Focus on the Noun, the Emotion, and Primary style.{{.Literary.Val}}

DO NOT PUT ANY TEXT IN THE IMAGE.`

After the algorithm selects the nine attributes from their respective databases, it proceeds to integrate these attributes into a predefined template to construct a comprehensive prompt. This process can be represented semi-mathematically as follows:

Let *Ai* denote the attribute selected for each category *i* where

*i* ∈ {Adverb, Adjective, Noun, Emotion, EmotionShort, Style, Color1, Color2, Orientation, Background, Literary}.

The template function *T* takes these attributes as input and produces a prompt *P*, defined by

*P* = *T*(*A*Adverb, *A*Adjective, *A*Noun, *A*Emotion, *A*Emotion, *A*ArtisticStyle, *A*Color1, *A*Color2, *A*Orientation, *A*Background, *A*Literary)

The function *T* essentially performs a string substitution where each placeholder in the promptTemplate is replaced with the value of its corresponding attribute. The result is a detailed prompt that guides the creative generation process, emphasizing the noun, emotion, and primary style, while explicitly excluding any textual elements from the final image. This structured approach ensures that the output closely aligns with the specified attributes, fostering a rich and coherent interpretation of the given descriptors.

Enhanced Prompt

Following the construction of the initial prompt *P* using the template function *T*, the algorithm engages an AI model *M* to generate an enhanced prompt *E*. This step can be mathematically described as:

*E*=*M*(*P*, *A*Literary)

Here, *M* represents the AI model's function, which takes two inputs: the prompt *P* created in the previous step and the attribute Literary *A*Literary, which specifies the literary style chosen from the databases. The model is instructed to refine *P* by incorporating the nuances of Literary *A*Literary, thereby producing an enhanced version of the prompt *E*.

The process aims to amplify the creative and stylistic elements of the prompt, ensuring that the final output *E* is not only in alignment with the initial specifications but also enriched with the depth and texture provided by the selected literary style. This enhancement step leverages the AI's understanding of literary styles to imbue the prompt with a specific tone, mood, or narrative technique, elevating the conceptual foundation laid by the initial attributes and their assembly within the template.

Image Generation

The culmination of the algorithmic process is the generation of an image based on the enhanced prompt *E*, facilitated by the AI model *G*. This final transformation can be described as follows:

*I* = *G*(*E*)

Where *I* represents the generated image, and *G* is the function executed by the AI model to interpret the enhanced prompt *E* and produce a visual representation. It's crucial to note that the process, up to and including the generation of the enhanced prompt *E*, is deterministic when provided with the same initial hexadecimal string and a static database. This determinism ensures repeatability and consistency of the output for identical inputs, as long as the databases used for attribute selection remain unchanged.

The databases, crucial for attribute selection, are meticulously maintained under version control and are published to The Unchained Index, a comprehensive repository designed to ensure transparency, accessibility, and integrity of the records it contains. The Unchained Index acts as an external reference, akin to a citation in an academic paper, providing a verifiable source for the databases used in the process. This setup guarantees that the deterministic nature of the process is preserved until the step of enhancing the prompt with the AI model *M*. However, the image generation step, executed by *G*, introduces a level of creativity and variability that makes each output unique, even if the enhanced prompt *E* remains the same.

Reference:

The Unchained Index: This citation refers to a hypothetical repository where the version-controlled databases are stored, ensuring that any given hexadecimal string mapped to this controlled dataset results in a predictable selection of attributes, up until the point of AI-enhanced prompt generation.

Important Note About Reproducibility

In the context of this algorithm, it's important to recognize that while each generated image *I* is unique, each pre-enhanced prompt *P* generated from the same initial hexadecimal string and static database is identical. This characteristic allows the pre-enhanced prompt *P* to function similarly to a "Blockie."

A Blockie is a visual representation, often colorful and patterned, uniquely associated with a specific Ethereum address or hash. It serves as a distinctive, cryptographic avatar for digital identities on the blockchain. Just as Blockies visually encode Ethereum addresses into identifiable icons, the pre-enhanced prompts *P* in this algorithm encode hexadecimal strings into distinct textual representations. Therefore, these prompts can be seen as textual counterparts to Blockies, offering a unique, reproducible output that represents the initial input string before the introduction of AI-enhanced variability. This parallel underscores the blend of uniqueness and consistency fundamental to the algorithm's design, leveraging the deterministic nature of digital identifiers to create reproducible yet creatively enriched outputs.

Conclusion

In summary, this paper presents a novel algorithm that transforms hexadecimal strings into visually and conceptually enriched prompts, culminating in the generation of unique images. Beginning with a deterministic process, the algorithm segments a hexadecimal string into parts, each mapped to a specific attribute from curated databases. These attributes are then assembled into a structured prompt through a template function. The resulting prompt is further refined by an AI to reflect a chosen literary style, enhancing its creative depth. The deterministic nature of the process up to the generation of the enhanced prompt ensures consistency and reproducibility, akin to the concept of Blockies in the Ethereum ecosystem. The final step diverges from determinism, as the AI generates a unique image based on the enhanced prompt, introducing variability and creativity into the output.

The databases, crucial to this process, are maintained with rigorous version control and are accessible through The Unchained Index, ensuring transparency and integrity. This methodological approach not only facilitates the creation of unique visual content but also embeds a layer of reproducible identity, similar to cryptographic avatars, within the generated prompts. Thus, the algorithm bridges deterministic data transformation with creative generation, offering a robust framework for digital creativity.

Appendix A will provide detailed examples illustrating the entire process from the initial hexadecimal string segmentation to the final image generation. Through these examples, we aim to demonstrate the algorithm's application, the role of deterministic and creative components, and the potential of combining structured data with artistic AI to produce unique digital artifacts.

Appendix B discusses the application of this idea to the realm of blockchain NFTs and other possible monetization strategies.

Appendix C lists the IPFS hashes of the version 1.0 databases used in this project.

Appendix A

0x3b161d57f482cd2dfbb626f0307ef92b3b094fce

A collage of images of a bird

Description automatically generated

Appendix B: Bridging Generative Art and NFTs on Ethereum

Transforming our innovative algorithm into a method for creating generative artworks minted as Non-Fungible Tokens (NFTs) on the Ethereum blockchain represents a groundbreaking leap in digital artistry and ownership. This approach not only leverages the inherent uniqueness of Ethereum addresses as a source for generating art but also embeds each piece with a verifiable digital provenance, making each artwork intrinsically tied to its creator and owner.

To implement this, the algorithm's output—each a unique piece of art generated from the hexadecimal string of an Ethereum address—can be directly minted as an NFT. This process ensures that the artwork is not just a digital asset but also a part of the Ethereum blockchain, carrying with it all the benefits of security, scarcity, and transferability that NFTs offer. Each piece becomes a collectible asset, whose value can appreciate not just from its artistic merit but also from its cryptographic uniqueness and the history of its ownership.

The excitement around this integration stems from the fusion of art and blockchain technology, creating a new paradigm for digital creators and collectors. Artists are empowered to explore new dimensions of creativity without sacrificing the uniqueness and ownership rights often lost in digital reproduction. Collectors, on the other hand, gain access to a world of exclusive, verifiable artworks that can serve as a digital legacy, each piece a testament to the intersection of creativity and technology.

By marrying the deterministic creativity of our algorithm with the Ethereum blockchain, we are not just creating art; we are redefining what it means to own and experience art in the digital age. This venture heralds a new era for artists and collectors alike, promising a future where art is not only seen and appreciated but also owned and traded on an unprecedented scale, all while being secured by the immutable ledger of the blockchain. The potential for innovation, expression, and investment makes this approach not just exciting but revolutionary, signaling a new chapter in the annals of art history.

Appendix C - The Attribute Datbases

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | nRecords | IPFS Hash | version |
| Adverb | 202023 | IFPSIP | v0.1.0 |
|  |  |  |  |
|  |  |  |  |

Other issues

Refer to this page to become a better prompt engineer: <https://platform.openai.com/docs/guides/prompt-engineering>.

There should be a whole section on the Templated Prompts. It’s a super interesting area of research.