Disposable AI Systems

Introduction

The building blocks of the SGS are Entity instances that each encapsulate their own HllSet. This is one-to-one mapping - one HllSet for each Entity instance. With the fixed parameter P in HllSet{P} definition we have finite number $2^{(64\ -\ p\ )^{2^p}}$ of possible different HllSets and as a result a finite number of Entity instances.

In a previous discussion on the entity.jl module, we explored how all operationally defined relationships between an entity instance and other instances form an integral part of the entity instance's definition. Each compound entity explicitly references its constituent entity arguments.

Another important feature of HllSets is their absence of analytically defined differentials, largely due to the static nature of a set, which remains unchanged. To enhance the adaptability of HllSets, we adopted a somewhat unconventional approach. We proposed that, under certain conditions, two or more sets can be regarded as instances of the same entity, even if their contents differ slightly.

We have determined that two or more HllSets are regarded as identical (represent the same entity) if they share the same entity ID. This marks a shift in our criteria for equality among HllSets from a focus on their physical representation to a conceptual framework. Consequently, HllSets that are logically equivalent may exhibit significant differences in their physical forms, particularly in comparison to those that are not logically equivalent. This concept of logical equivalence adds a new dimension to the traditional notion of equivalence, which is based solely on the equality of set contents.

This approach allows us to handle a series of different states of the HllSets and apply measurements that can approximately be regarded as differential and gradient analyses.

If we consider HllSets as mapping domains for real datasets, we must acknowledge that the co-domain represented by these datasets is inherently constrained by the datasets themselves.

To advance our discussion, it is essential to revisit the fundamentals of transforming a real dataset into HllSets. In this process, each element of the real dataset is converted into an integer hash value. These integers are then utilized to construct an HllSet. Whenever we attempt to draw inferences about future states from the current state of the SGS, we are navigating between a finite domain (the real data) and a co-domain (the HllSet representation).

This specifically entails:

1. The `**advance**` function, incorporated within the `**entity.jl**` module, is designed to facilitate inference. This function concludes its operation by invoking `**create_new(b.hll)**`, a feature yet to be implemented. Upon completion, it is expected to generate two types of HllSets: `**n**` (**new**) and `**r**` (**retained**).
    a. The creation of the `**n**` HllSet adheres to specific constraints: it should be composed of elements from `**b.hll**`, augmented with elements sourced from other entity instances, as well as additional elements not currently associated with any existing entity instances (‘**unkn**’).
    b. Conversely, the `**r**` HllSet is exclusively constructed using elements derived from `**b.hll**`. This ensures that the `**r**` HllSet remains a pure subset of the original HllSet from which it was derived.

```python
# ::Colon is placeholder for missing argument
# b::Entity{P} is the current state of the entity instace b
# So, we are this variant of the function to project current state
# of the entity instance to the future
function advance(::Colon; b::Entity{P}) where {P}
       # Create a new empty set
       a =create_new(b.hll)       # Placeholder for the function that creats new
                                  # unknown Entity instance
       d, r, n = diff(a, b)
       op_result = Operation(advance, (d, r, n))
       # calculate the gradient for the advance operation as
       # the number of elements in the a set
       grad_res = HllSets.count(a.hll)  # This is the simplest way to calculate
                                        # the gradient
       # Create updated version of the entity
       return Entity{P}(hll_res; grad=grad_res, op=op_result)
end

function create_new(hll::HllSet{P}) where {P}
       n_hll = HllSets.HllSet{P}()
       r_hll = HllSets.HllSet{P}()
       # TODO: implemnet code to build n_hll and r_hll

       return HllSets.union(n_hll, r_hll)
end
```

2. If the future state of the entity instance can be represented as a composition of other entities integrated through set operations (such as union, intersection, etc.), we can attain the final result by merging various instances. In this scenario, the objective is to discover an optimal composition of the input instances.

With this we can move to the next chapter.

Training epochs and a self reproduction loop

The training process in modern AI systems feels somewhat unnatural to me. It presents a fascinating resolution to the classic "catch-22" dilemma. On one hand, you cannot begin training until you have gathered sufficient information and knowledge. On the other hand, you cannot utilize an AI system to collect this information prior to training. As a result, people are stepping in to bridge this gap by generating the necessary information, which ultimately feels counterintuitive.

The process of training models through successive epochs bears a certain resemblance to the self-replication cycle. However, there are fundamental differences. The SGS system is designed to commence learning instantly upon activation and immediately begin processing the incoming data.

"Will it work?" This is a valid question, and the answer is likely yes. However, a more complex inquiry arises: how long can it function sustainably?

Even more challenging is the third question: can the system learn if its only source of knowledge is the data we provide?

Understanding this helps simplify the response to the second question.

The system will continue to operate effectively as long as it has the capability to learn.

What happens if a self-reproductive system loses its ability to learn? Naturally, the system will cease to function and will be discarded. This is an inherent consequence of all self-reproductive systems—they must be decommissioned to make space for new generations.

In the next chapters we will discuss how we can use the self-reproduction loop learning process.

What’s is an objective of training AI model

The ultimate objective of training an AI model is to establish weighted relationships among tokens that symbolize aspects of reality. This task is inherently complex and involves intricate combinatorial challenges. The difficulty compounds when we aim to assign varying weights for different contexts, sub-contexts, and types of reality.

Furthermore, in light of the necessity for a sustainable framework—given that retraining entails considerable expense—we are confronted with a substantial workload, which in turn reinforces our job security. Although the project is underway, there is no clear end in sight. The monolith continues to expand, seemingly without any strategy to manage its increasing complexity.

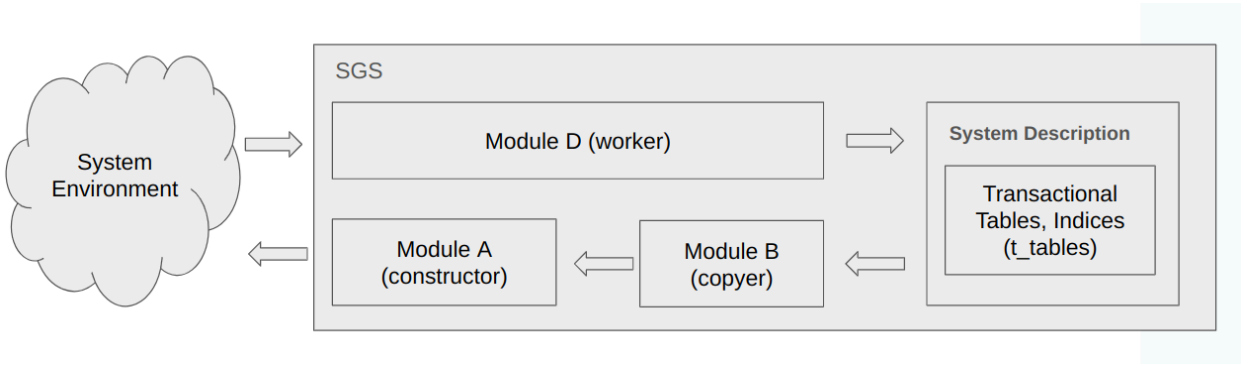
A key conclusion is that current AI models thoroughly outline all elements and their interconnections for every possible scenario presented in each request.

This prompts an inevitable question: **Is it necessary to include all these details at all times? Personally, I find this approach excessive and somewhat unnatural.**

In my view, any system, including AI systems, should be built on a streamlined core structure that ensures its sustainable operation. This core should possess the capability to expand by utilizing built-in operations to create extended sub-structures as needed.

Self-reproductive loop as lifelong learning model

Below is a diagram from one of my previous posts [1]. It shows a self-reproductive loop in terms of John von Neumann self reproducing automata.



Module **D** obtains inputs from the **System Environment** and records these inputs by generating records in the "**transaction**" index. Simultaneously, Module **A**, with assistance from Module **B** (the copier), retrieves these references from the "**transaction**" index. It then processes these references by engaging the appropriate processors and subsequently uploads the processed data back into the System.

It is crucial to note that SGS never directly sends incoming data to the System. Instead, it first segregates all incoming data logically into a staging area using the references in the "transaction" index.

This approach enables us to achieve several key objectives:

1. **Clear Distinction**: We establish a clear separation between existing data in the system and newly acquired data.
2. **Comprehensive Control**: We gain complete oversight over the processing of new data, allowing us to track completed tasks and pending work. This also supports parallel processing and facilitates recovery from failures.
3. **Data Type Isolation**: We can effectively isolate unsupported data types.
4. **Adherence to Protocol**: We strictly follow the self-reproduction flow.

To submit newly acquired data, we utilize the commit command. Within the Self Generative System (SGS), each entity instance is classified according to one of three primary commit statuses, which are essential for tracking modifications:

5. **Head**: This status indicates that the entity instance reflects the most recent modification.
6. **Tail**: An instance marked with this status represents a prior modification, signifying that it is not the latest version.
7. **Deleted**: This status is assigned to instances that have been flagged as removed from the system.

Each commit in the system is associated with a unique "**commit forest**", represented through a distinct matrix. For every commit, there is a corresponding matrix. There is no need for concern; when the status of the previous version is switched to 'tail,' it simply denotes that it has become a historical entry.

The updating process is systematic and involves several key steps:

1. Creating a new version of the item to be updated.
2. Implementing the necessary modifications to this new version.
3. Saving these changes.
4. Establishing a connection between the new version and the previous version of the item.

The SGS self-reproductive loop enhances the SGS data structure by creating new HllSets that explicitly define the differences between the previous and current states of each modified entity. As previously mentioned, this differentiation is represented by three new HllSets:

1. An HllSet representing tokens **deleted** from the previous state of the entity;
2. An HllSet representing tokens **retained** in the current state of the entity;
3. An HllSet representing **new** tokens added to the current state of the entity.

Important to note that these new HllSets are introduced into the system along with their corresponding relationships.

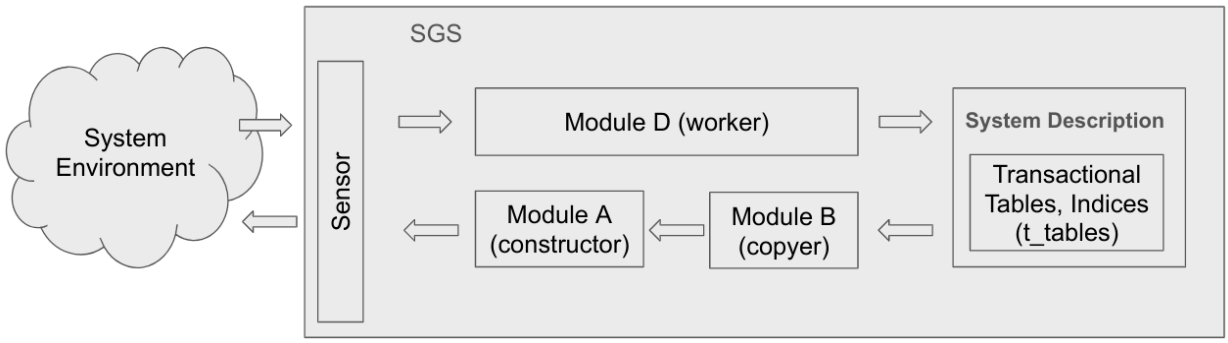
It is anticipated that any engagement with machine learning (ML) or neural networks often introduces multidimensional Cartesian diagrams. This is understandable, as both ML and neural networks fundamentally rely on linear algebra. However, the scenario changes with HllSet-based systems.

HllSet itself is dimensionless, and the SGS evolution generates a **one-dimensional sequence** representing the evolving states of SGS entities.

So, do we face a significant challenge? To some extent, yes. We must devise a solution to this issue. Fortunately, we do not need to search far—the answer lies within us. Although our brain does not have dimensions in the Cartesian sense, it is adept at processing spatially dependent entities. We will explore this topic further in the next chapter.

Multidimensional spatial entities in SGS

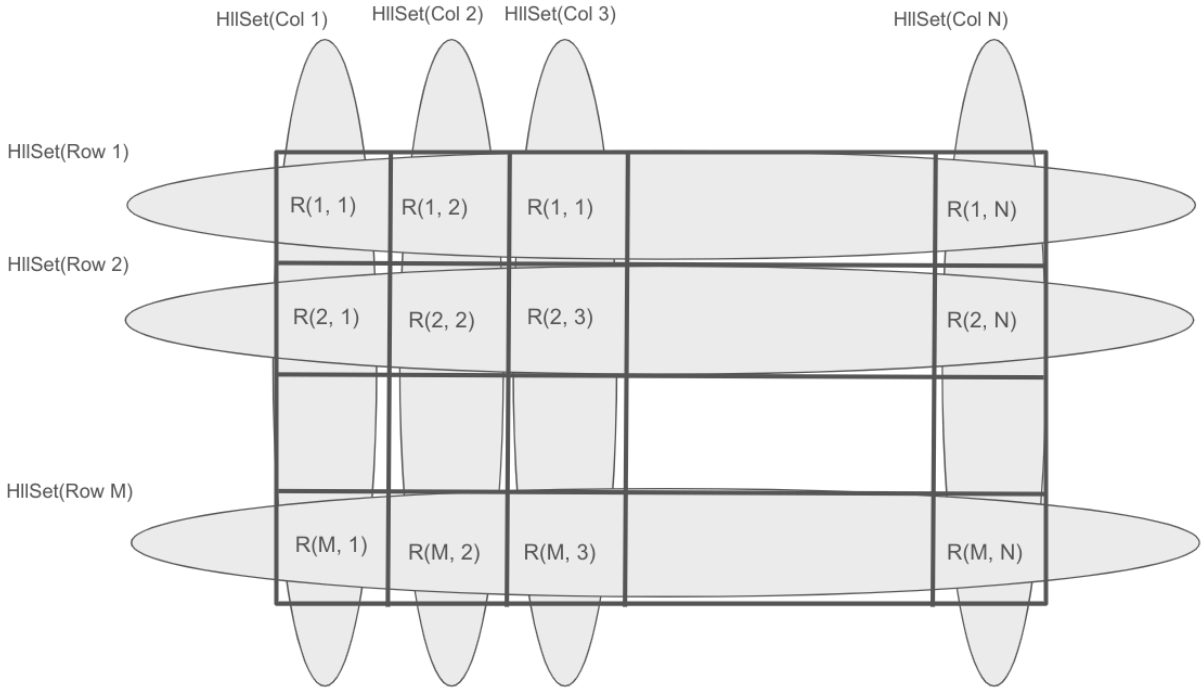
We updated the diagram from the previous chapter a little bit by adding a sensor (perceptron). We can think about it as a digital camera sensor. The pixels in this sensor are arranged in rows and columns (it is two dimensions) and each pixel has multiple states that depend on light intensity and color mask (R - red; B - blue; G - green), that is the third dimension.



In the image sensor, each pixel is identifiable by its row and column coordinates, which can be integrated into the pixel's value. Within the context of HllSets, these values serve as unique tokens representing individual pixels. By this definition, tokens assigned to distinct pixels are inherently different.

To summarize our approach: once pixels are encoded into tokens and transferred to the HllSets, the original order of the pixels is lost. However, the potential to reconstruct their original sequence remains, as each token embeds the pixel's positional information.

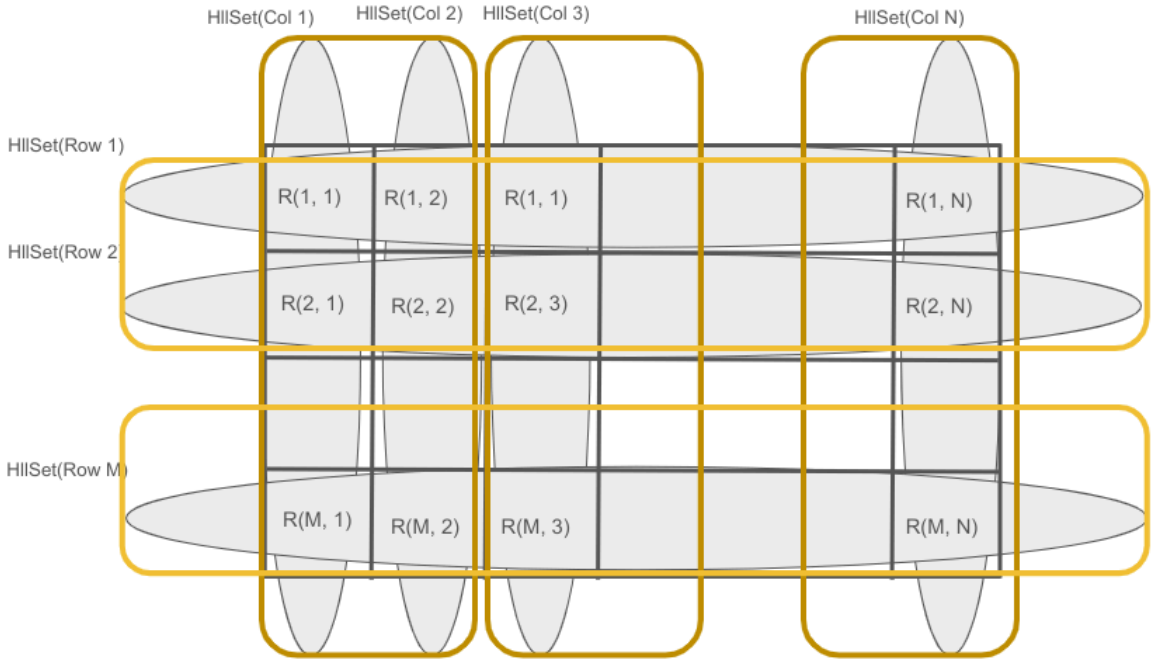
In our previous post [2] where we discussed HllSet relational algebra, we proposed the projection operation. This operation was defined for two collections of HllSets and allowed us to project one collection of HllSet onto another one. In the picture below we can see this operation applied to the tabular data.



This diagram depicts the projection of M rows onto N columns, where the intersection of row i and column j contains an HllSet representing the pixel R(i, j).

In dealing with tabular data, such as an image sensor which organizes pixels into a table format, we do not generate an HllSet for each individual pixel. Instead, we create an HllSet for each row and each column. These HllSets implicitly contain the pixel information. To extract this information, one simply needs to intersect the HllSet of a specific row with that of a specific column.

This methodology also enables additional image processing capabilities. For instance, to reduce the image resolution, one can group rows and columns and then perform a union operation on each group. The accompanying illustration demonstrates this process by grouping rows and columns in pairs.



After this grouping, the number of rows will be reduced to (M / 2), and the number of columns will be reduced to (N / 2). Consequently, the intersection of a new row and new column will return a new pixel that will be represented by four original pixels.

The content of text differs significantly from that of images, necessitating distinct processing approaches. One viable solution could involve leveraging the selective capabilities of HllSet algebra combined with the robust power of generative large language models. The HllSet algebra would generate a collection of unordered tokens that meet specific user search criteria, while ChatGPT or a similar AI model would then synthesize these tokens into coherent and digestible text.

Summary

Here are some insights derived from the document regarding the training processes of contemporary AI systems and the potential of Self-Generative Systems (SGS).

Challenges in the Training Process

The initial phases of AI training present a significant challenge, often likened to a "catch-22" scenario. A substantial dataset is necessary to kickstart the training process, yet the AI system is unable to assist in gathering this essential data until it has undergone some training. This paradox necessitates human intervention to generate the critical initial data, which may seem counterintuitive.

Innovations in Model Training

Traditional training methods for AI models typically involve multiple epochs, resembling a cycle of self-replication. However, the Self-Generative Systems (SGS) mark a transformative advancement in this field. These systems initiate their learning process immediately upon activation, processing incoming data in real-time and converting it into actionable knowledge for future applications.

Sustainability and Functionality

A crucial consideration for these systems is sustainability. While the immediate functionality of SGS appears promising, their long-term effectiveness hinges on their continuous ability to learn and adapt. If an SGS loses this capability, it will cease to function effectively and will need to be decommissioned in favor of newer generations. These successors are direct descendants of the currently operational models and, with proper knowledge transfer, can begin their development from a more advanced baseline.

Dependence on Provided Data

Another critical aspect to consider is the sustainability of these systems' knowledge acquisition, which is heavily reliant on the data provided to them. The longevity of their functionality is directly linked to their learning capabilities. Feeding data to an SGS can be likened to educating a child—new information should be delivered in digestible portions, and the content should align with the specific domain targeted by the SGS.

Strategic Implementation and Limitations

The document outlines a novel approach to the development of General AI, highlighting both the potential and limitations of SGS. It underscores the necessity for strategic planning to address these limitations and fully leverage the advantages of SGS, particularly their immediate learning capabilities upon activation.

Conclusion

In summary, while current training methodologies for AI systems have inherent limitations, the emergence of Self-Generative Systems offers a promising alternative with the potential to revolutionize the field. It is essential to continually assess these systems for sustainable functionality and adapt our strategies accordingly.

Should you wish to explore these insights further or discuss them in greater detail, please feel free to reach out.

References

1. https://www.linkedin.com/posts/alex-mylnikov-5b037620_hllset-commit-and-self-reproductive-system-activity-7203144734293250048-WgM2?utm_source=share&utm_medium=member_desktop

2. https://www.linkedin.com/posts/alex-mylnikov-5b037620_hllset-relational-algebra-activity-7199801896079945728-4_bI?utm_source=share&utm_medium=member_desktop