Chimera: Evolving Software

MILESTONE 1

Repo link: https://github.com/ChimeraMetta/Chimera

The first milestone for the Chimera project has focused on what we've called the 'core donor system', which is a slightly pretentious way of saying we've set up all the infrastructure required to build the rest of the coming milestones out and release a full demo of all the system's capabilities.

Because this was also essentially a discovery milestone, experimenting with the MeTTa ecosystem, there have also been some shifts in the approach to this infrastructure setup as we've proceeded:

1. Due to dependencies for PyPy 3.8 for the `hyperon` library, the original goal of an "SDK" will not be possible when working to bridge between MeTTa and a 'regular' language like Python. As such, we've pivoted to focusing on a CLI tool instead

2. In the design of the system, I have realised that there needs to be more focus on the relationship between symbolic systems like MeTTa and statistical systems like LLMs than originally proposed. More on this in following sections

How the Concept has Played Out

In the development of Chimera, I've realised that what MeTTa and the Atomspace are excellent at is deduction via relational mapping/ontology. For the system, I've started with:

1. ***Python Ontology:*** Knowledge about how Python code components (functions, classes, imports, etc) are related to one another, both directly and transitively. An ontology for how Python as a language works

2. ***Codebase Ontology:*** Knowledge about *this* specific Python code being worked on right now. This ontology is generated on the fly by Chimera by ingesting and then converting the code into a MeTTa format, supported by the Python ontology base

This is a great foundation to develop further from, but we also wanted to provide something demonstrable for the first milestone. By running `chimera summary file.py`, you can see this foundation in action on your local Python file. The command will go through your file and use MeTTa to extract meaningful data about its ontology, providing this to you in a range of different considerations (eg. code complexity, class relationships, etc)

While this is a good first step, it is not enough.

Automated Proof Generation

The other component required to consider the core donor system complete is, of course, the donor system itself. While code ontology is core and necessary, if we want to expand Chimera to being able to heal from errors and improve code, it needs to semantically understand the intent of the code and then provide alternatives to run and test.

We needed a proof system for code.

Based on both prior experience as well as newer research by others such as in this paper, we developed a proof system for code that comprises:

1. Pre-conditions
2. Post-conditions
3. Loop invariants
4. Other minor components such as bounding, null and error checks

The above has been embodied in practice as a conversion of Python code into a JSON intermediate representation of proof components (generated by LLM) that are then converted once more into MeTTa representation.

With the MeTTa representation generated, we now have something to compare new alternatives to. Now, an LLM can generate alternatives to a broken, complex, or otherwise undesirable function *and* MeTTa can verify that the alternative will work from a proof system.

The initial implementation of this for demo purposes has been promising. We've started with a simple command to reduce the complexity of functions in your code that MeTTa/Chimera deems to be 'too complex'. You can run this as `chimera analyze file.py --api_key=OPENAI_API_KEY`, where supplying an OpenAI API key will allow Chimera to generate alternative functions and suggest ones that best reduce the complexity.

```
=== Best Alternative ===
Strategy: semantically_equivalent
Properties preserved: True

Code:
def analyze_text(text: str) -> Dict[str, Any]:
    """Perform advanced text analysis with multiple operations and loops.
    This function has been deliberately made complex.
    """
    # PRECONDITION: text is not None
    if text is None or text == "":
        return {"empty": True}

    # Word frequency analysis using a dictionary comprehension
    words = re.findall(r'\b\w+\b', text.lower())
    word_freq = {word: words.count(word) for word in set(words)}  # LOOP INVARIANT: word in words
implies word_freq[word] ≥ 1

// ...

To use this optimized implementation:
1. Create directory: optimized
2. Save to: optimized/test_file.py
3. Replace the original function with this optimized version
```

MeTTa Only!

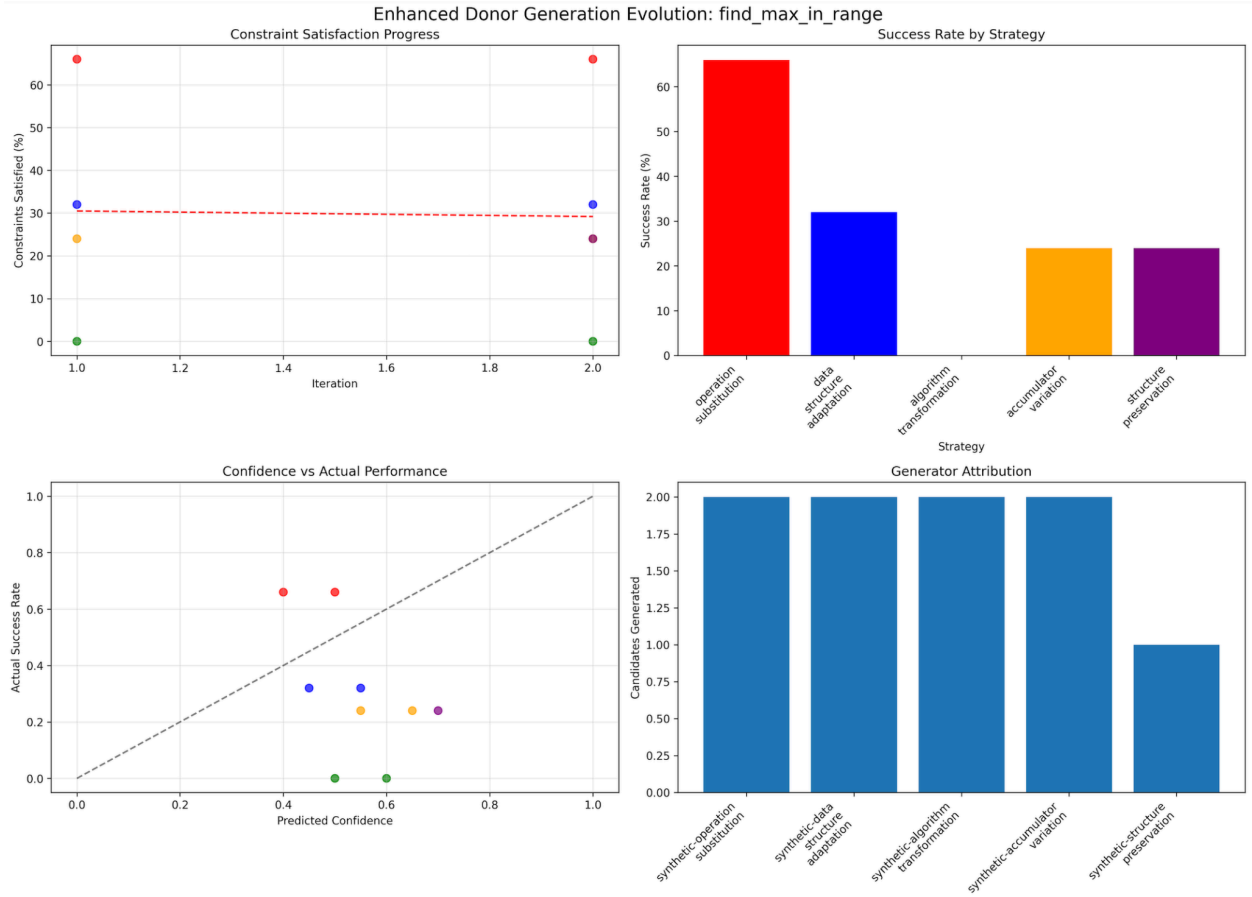
As per the original milestone descriptions, we don't want a system that only works with LLMs for generation. It's important to demonstrate that MeTTa alone is capable of generating solutions that evolve over time, so we've added 2 commands to the CLI tool:

1. **Generate:** Running `chimera generate my_file.py` will generate MeTTa-only solutions to donor generation. This uses a relatively complex modular MeTTa donor generator system that combines multiple different strategies in order to produce candidates. For now this is simply aiming for parity with original functions, but will develop to include evolution toward specific goals in future milestones:

2. **Visualize:** While it isn't particularly useful for production use of Chimera as a tool, it can still be interesting to see how this generator system is evolving candidates over time. Running `chimera visualize .` will produce a series of progress images for a single demo function, a simple `find_max_in_range`. The plots produced are rudimentary for now, and simply serve to show how the core donor system could work to generate new candidates in future. Without specified goals (such as "fix error" or "improve feature handling") which require further development, this feature shows what's possible in the system's evolutionary approach.

In addition, without goals outside of parity and with a currently limited internal Atomspace set, there is no guarantee that the current system can *always* converge to a perfect solution using

MeTTa alone. It will be interesting to explore the combination of MeTTa evolution and LLM generation for production use in further milestones.



Something Extra

I've realised while working on the updated milestone that the import/export feature of Chimera can be broken out into its own module for others to use in their MeTTa projects. The ability to save and share Atomspaces conveniently will be useful not only for Chimera's growth as a project, but for the development of other MeTTa projects as well:

I have therefore begun work on a separate module called Janus, which will be a small, standalone Python module for handling the import and export of Atomspaces. I'll release this for general use in a future milestone.

Next Steps

Although this is interesting in terms of theory and the demo can display these concepts, in order to generate something that truly pushes toward something more AGI-like and does more than simply an LLM could do, the next milestone steps will be designed to show how this foundation can be used to let an agent handle things autonomously.

Current LLMs are powerful (and getting better by the day) but they're still not capable of full autonomy due to the mismatch between statistical generation and the hard requirements of proofs. I think MeTTa can resolve this mismatch, and an autonomous healer system in the next milestone will be a much stronger demo of this capability.

If anyone on the DeepFunding team has any questions or concerns, please feel free to reach out. Further information on the testing and running these demos can be found in the Chimera repo here: https://github.com/ChimeraMetta/Chimera/tree/main