SUMMARY OF DESIGN

There are two main components necessary: an online REST API described above and a batch training job to be run at regular intervals. Presently all server software is written in Python, utilizing Flask/mod_wsge on Apache httpd.

REST API PROCESSING

First, the REST API is described. For all the transactions, the transaction is checked to make sure the REST submitter has the valid credentials, and if applicable the case number must be included.

1) If the user submits a /createcase call then a case is initialized in a MySQL database, and the case number is returned.

2) If a /sys1_proposition is submitted, it is added to the case in the MySQL database. Checks are made to ensure proper ordering. If a transaction contains “appearance” fields, it is deemed to be level 0 in all cases. A single level 1 judgment can only be inserted after level 0 cases with the same subject. Only one level 2 judgment is acceptable and it must follow a level 1 judgment. The server replies as needed (eg., “judgment added”, condition 200, “error out of order,” etc).

3) If a /sys2_proposition is submitted, it is added to the case in the MySQL database. Checks are made to ensure proper ordering. It cannot contain “appearance” fields (see above). A single level 1, system 2 judgment can only be inserted after level 0, system 1 cases with the same subject. Only one level 2 judgment is acceptable and it must follow a level 1 judgment. The server does not evalute a case until a sys2_realistic call is made below.

4) /sys2_realistic will trigger a call to the AI kernel (see below). After the kernel processes the whole case, the Apache server responds with either “realistic” or “unrealistic,” meaning that the submitted case is consistent with past cases or it isn’t. If any new judgments are derived these are returned as well. Note that a case can only be determined as “realistic” if all “desired” terms appear in respective judgments. Desired terms can be included in any sys2_proposition call. They are not required.

5) If a /retract_that is submitted, the most recent judgment in the MySQL database for the case number passed is deleted.

6) If a /traincase call is made, the MySQL database is updated with the training flag set to true (initialized false).

AI KERNEL

If one were to summarize the design strategy in short form it would be “old AI symbolic and modern AI deep learning chaining in parallel.” Thus, this is a neuro-symbolic strategy.

When a sys2_realistic REST transaction is submitted, this means that the developer’s software is unable to proceed on its own. The kernel inputs the entire case including all system 1 and system 2 judgments that have been posted using respective REST transactions as above (with any keyword absolutely required in a specific system 2 judgment specified as “desired” and any associated essence and appearance data as described above included if needed). The basic function of the kernel is to evaluate the totality of judgments submitted, system 1 and 2, as a single, coherent, and unified case and try to determine whether or not the case as a whole is essentially "realistic."

The basic structure of AI kernel is similar to – yet has evolved from – that described in the book "Approximation Zero," published 2013 by Michael Archbold, ISBN 0615882811.

The book (2013) describes a three part ontology: the "What," "How," and "Toolbox." The "what" in the book basically consists of propositions that can act as a focal point in, and include, an overall idea. An idea is roughly an integration of multiple propositions in a case into a single understood totality, the whole works being the what. The "how," is basically the means needed to create the "what." Fundamentally the "how" is composed of a "megafunction" and an "oracle." The megafunction (so called because it should be able to act – ultimately – on whatever judgment the oracle commands) inputs judgments from the oracle and outputs an understanding for the oracle, and the oracle inputs understandings from the megafunction and outputs a single selected judgment then input to the megafunction for execution. (An "understanding" was never adequately defined in the book, however it roughly meant whatever is needed by the oracle to choose one judgment from a set of possible judgments. This was unsatisfying to the author but left open.) The megafunction and oracle work in tandem to produce the what. The what is (again) roughly an understood idea with a propositional focal point. There is always a propositional focal point within an overall idea.

As described in the book, the process starts with a "main idea" being issued to the AI – ie., the user submits a case for evaluation. Upon receiving a main idea the oracle and megafunction interact until the "desired ends" of the main idea are "objectively realized." These last two quoted expressions are drawn from the Hegelian ontology. The "toolbox" mentioned above consists of discrete, abstract rules which are utilized by the megafunction in generating the understanding for the oracle.

The present code (2022) is structured in an object oriented fashion using the book's general scheme. The kernel is composed of an oracle and a megafunction (both Python functions). The "how" as sketched above became Python executable code and the "what" sketched above became Python data structures. The "toolbox" consists of distillations of normalized, compressed, and trained models based upon all prior cases.

When a sys2_realistic transaction is processed, the oracle (again, part of the kernel) initially chooses the last judgment that was submitted for the case (typically with /sys2_proposition) which is assumed to be the main idea. The entire case is then evaluated in turn, starting with the main idea, until the oracle decides that the main idea is realistic or unrealistic based upon the totality of the case.

Every judgment submitted in a case must be shown to be realistic to the oracle. A system 1 judgment, as mentioned above, is simply taken to be true (thus also "realistic") by the kernel. However any system 2 judgment must be evaluated.

The core of the system 2 judgment evaluation starts with the classic Aristotelian, categorical syllogism. The classic syllogistic example is:

> Socrates is a man
> all men are mortal

Socrates is mortal

The second premise, here a universal, is typically induced from an adequate number of particular cases.

A 21st century version of the classic syllogism, however, serves as the core for the kernel's evaluation.

Given some input judgment, the kernel tries to resolve a judgment to what is fundamentally, at bottom, a syllogism, and if and when the syllogism chains terms (Socrates → man → mortal), the judgment is deemed realistic. If the "Socrates is mortal" case were submitted to the kernel for training, "Socrates is a man" would be the first judgment, followed by the "Socrates is mortal" judgment, but the server would learn "all men are mortal." There is no need to submit the "all men are mortal" major premise. Basically the "all men are mortal" premise learning is the object of deep learning, the benefit being that a great number of variations are possible (although obviously the context of mortality is rigid).

In addition, more than one minor premise is allowed. So in the above example, suppose other statements are made about Socrates: Socrates is a philosopher, Socrates is a stonemason, and so forth. The "all men are mortal" premise, the major, is really an ideal location to use deep learning instead of an old AI term chain. Omit the "all men are mortal" premise entirely. Mentally insert a CNN (convolutional neural network) there instead to do the same thing: the major premise is induced from particular prior cases, and serves as the bridge to the conclusion. The conclusion could become "Socrates is mortal, is skilled with his hands, and is wise" if multiple minor premise judgments are submitted as above.

So, we sort of crush a mass of syllogisms into one: call it a "mega-syllogism." This structure gives a great deal of latitude to any syllogism. Now we have one syllogistic structure that can handle a wide variety of propositions about Socrates.

If we take all possible statements about Socrates we can ever funnel through the CNN we can call the resulting functionality an "idea" – if we define an idea as an integration of possible or actual propositions; thus, in a manner of speaking an "idea" about Socrates is learned.

In addition to the CNN functionality, there is also an old fashioned term chain. In the above example, in order for "Socrates is skilled with bricks" to be realistic, the server will have to see a proposition "Socrates is a stonemason" directly followed by "Socrates is skilled with bricks" and learn the connection of "being a stonemason means skilled with bricks" (a de facto major premise) using offline training.

In this way the inference is held together **both** with a CNN and with old fashioned term chaining. Thus for a system 2 judgment to be deemed realistic it has to pass this two-part test.

A significant problem with all of AI is known as the combinatorial explosion. The problem is that there exists a seemingly infinite number of outcomes to most real world situations (save those tightly constrained) making it practically impossible to pre-code every eventuality. It is the *context* of the situation which narrows down the space of possible outcomes, making for a hopefully tractable search.

Just processing a plain syllogism, as above, is not enough to create a tractable context if the intent is generality. So, to create added context, the above mentioned fields of "appearance" and "essence" in the API are utilized by the kernel to narrow down and refine the context. These fields are also based on the Hegelian ontology. These two fields are typically present in level 0 judgments which is always taken as true (system 1). The kernel considers level 0 judgments to be de facto minor premises in a syllogism.

The kernel gains context from appearance (what the input looks like) and essence (significant considerations used to make a judgment).

This added information narrows the context; however, this is taken a step further. In the Hegelian ontology something is not understood by what it is per se, but what it is like and unlike. A “reflection” of an apple might contain a like set of apples of similar color, and an unlike set of apples dissimilar color. Holding both sets in the mind in a single reflection concurrently gives an understanding of some apple in particular by comparison to what *this apple* is like and unlike. In psychology, this is basically known as a concept built from exemplars.

The intent of the present design is reduce the combinatorial explosion by creating such reflections alongside each judgment, the reflections adding crucial context to the syllogism, and feed these into the CNN for pattern matching along with the usual premises as illustrated above.

Thus when a judgment is processed by the kernel, if there exists passed term keywords (which need not be complete sentences) describing appearance and/or essence, these keywords are used to build reflections that contain likenesses (and unlikenesses) to some other, known aspect of appearance and/or essence.

So, to take our Socrates example: if keywords describing Socrates’ appearance and/or essence relevant to the case at hand are passed concurrently with the judgment “Socrates is a man,” these are used to build reflections not unlike exemplar concepts based on contextually relevant essence and appearance keywords that have accompanied the judgment. This enhances context.

The reflections constructed are typical NLP embeddings. The likeness of terms needed for reflection to add context, as described above, are based on the well-known Word2Vec scheme. Using Word2Vec training, essence and appearance terms that regularly appear together are assigned vectors close to one another in vector space.

If an input judgment contains keyword terms in essence and/or appearance, some of these are selected by the kernel for reflection. Not all passed terms are chosen for reflection – only those terms (in appearance and essence) that seem to be the most relevant given the context are chosen for reflection. The kernel tries to favor terms that are most closely associated and relevant with the judgments that have been passed. The judgment is the source of contextual information. In our Socrates example, the kernel would start with “Socrates, man, mortal” terms, then it would review past cases to ascertain which essence and appearance terms seem to occur the most. If certain appearance terms seem to occur a lot in past cases, given “Socrates, man, mortal,” and the same are in the present case, these are chosen for reflection.

However, if a more particular judgment is passed, such as “Socrates is a man lacking sandals” then the kernel selects terms from essence and appearance that seem to be relevant to “lacking sandals” as well, the decision made by examining prior similar cases.

Once the kernel selects the most contextually applicable terms, a reflection is built. The reflection is composed of about half a dozen (a variable parameter) embeddings of terms. The reflected embeddings are for those terms most like the selected applicable term(s) given Word2Vec training based on all prior cases. In this fashion we create a substructure analogous to an exemplar concept, or an Hegelian reflection.

There is a deliberate attempt to build something akin to mental concepts with reflections, although the similarity is merely analogous. A reflection of “apple” could give “orange, lemon, grapefruit,” or even “lettuce,” etc. Most people are familiar with the way Word2Vec associates terms. It depends upon the embedding created by Word2Vec training of prior trained system 1 and system 2 cases. It is hoped that the reflection described herein is akin to a mental concept, and thus closer to the psychological realm. It is the *structure* of mind that we are always striving toward. These reflections are intended, while admittedly somewhat crude, to narrow the context, to be tractable, and fed into a CNN to ameliorate the combinatorial explosion.

When we chain a syllogism, in summary, we are showing that the submitted judgment is realistic given prior cases. We are using multiple means – an old fashioned term chain along with a CNN that relies on reflected embeddings of the most contextually relevant terms. Simply put, if the term chain and the CNN succeed in showing that the premises support the judgment (the conclusion), and if “desired” terms are in the conclusion (if included), the judgment is realistic, and that is returned by the kernel to the API caller.

There is only a single level 2 judgment allowed per case. The structure of the syllogism described above, symbolic and CNN processing, is also imposed on the level 2 judgment. The difference in processing for the kernel is that the minor premises are all the level 1 judgments. The result is that the level 2 judgment binds together all of the level 1 judgments, thus achieving something akin to a single integration of the case. Thus level 1 judgments use level 0 judgments as minor premises in a syllogistic scheme, and the level 2 judgments use level 1 judgments accordingly. There is a hierarchy, but it isn’t much.

It is the job of the level 2 “final judgment” to reconcile all subordinate judgments in the overall context.

If the “?” character is submitted as a level 1 or level 2 judgment, the kernel will return “realistic” along with the derived judgment or “unrealistic” depending on the outcome. An ellipsis placed after part of a judgment will trigger completion of a partial judgment.

The design of the kernel is such that the oracle selects a judgment and then the kernel shifts control to the megafunction. The megafunction executes concepts that act on the data structures and objects. A concept, for example, could create a reflection for a particular term as discussed above. The oracle will try to show that each submitted system 2 judgment in the case is realistic. To do this it may need to branch. For example, to resolve a level 2 judgment, it needs to first resolve each related level 1 judgment.

Control shifts back and forth between the oracle and megafunction until a decision is reached – if the case is realistic or not realistic.

Most of the inspiration for the architecture stems from the ontology of Hegel as presented in his obfuscating “Science of Logic” with a desire to fuse it with deep learning. A similar notion of using three levels as a crucial part of an AGI structure was presented in the paper “A Metamodel and Framework For AGI” by Latapie and Kilic, the former being heavily influenced by the works of Korzybski.

BATCH (OFFLINE) TRAINING

All submitted judgments are saved for training and as a log.

There are steps to build the Word2Vec embeddings, create the CNN’s model, and create the old AI style term chain (a Python dictionary).

ADVANTAGES OF THE AM ENGINE

1) Software may be able to decide previously undecidable cases, thus becoming more general.

2) While this is a loosely coupled, parallel integration, of an AI server with conventional software, the structure can be evolved to a more tightly coupled integration resulting in ever increasing functional generality.

3) The structure is based upon the relatively simple syllogism enhanced with the power of deep learning.

4) Whereas the present syllogism scheme fires based primarily upon CNN networks, the design can be modified to use many different methods including transformers.

5) It is hypothesized that the design can handle temporal reasoning and other aspects of Hegel’s ontology beyond his categories included in this release. This release includes most prominently the “appearance,” “illusory being” (named “essence”), and “desired ends” categories but most other categories could be referenced. The code does presently contain more fields in the ontology but they aren’t referenced. Additionally there are three possible time modes per belief: past, present, future. However, only the present mode of time works.