Cover Letter

I have a specialization in Machine Learning and other related fields.

Also, I have developed a new **global optimization method that is better than Multistart** (without basins of attraction).

(https://medium.com/@pushkarevvaleriyandreevich/gradient-descent-that-we-must-have-5a4542e218a0

Read from What to do (with gradient descent)?)

That method performs 2-3 times better than Multistart. And if I add basins of attractions or other optimization techniques, I will outperform even GlobalSearch and others.

And I want to build an Ideal Optimization Pipeline.

First, simply running global optimization is cost-sensitive, we have too many solutions, and checking all of them is almost impossible.

But we can:

1) **Add some approximations of a solution** (with a neural net that is trained to act like a bloom filter or even an SVM (that can act like a set of rules)).

2) **Make many bloom filters** to reduce the possible solution space.

3) Run developed global optimization within that area (that is already 2-3 times faster)

And that will work better than GlobalSearch from Matlab (and other algorithms):

1) because we have not only basins of attractions but also approximations based on neural nets

2) We have a better global optimization algorithm beneath all of that

3) We can even add some NLU features to make more conditions/filters

Also, I must transfer all of that to OpenCL/Tensorflow to make it usable.

And I can solve any problem, even within computational engineering, faster than others.

What about NLU?

Well, I have a modern NLU approach that covers almost all needs of Automatic Design and can be verified and explained.

Heuristics, constrains and rules can greatly decrease the number of possible solutions during automatic design.

Also, nobody will build a model(s) of objects in Diagram Editor or so on (try to build 2-3 things with more than 20 parts) on a regular basis.

First,

What about objects and the properties of objects?

Again, there are many benchmarks for Q&A. And nowadays, neural networks with a 30 mb size can find answers better than humans.

And with Q&A it's possible to get almost any information about objects (what is the **color** of **object_name**).

It takes two modules to do that - the first module projects objects to the syntax tree:

{book {num_of_pages = 10, color = white, author=null}} - white book with 10 pages.

The second one - builds a question to properties

{book {num_of_pages = 10, color = white, author=null}} - Who is the author of a white book with 10 pages?

Note that we can ask many questions, and even incorrect ones (that will lead us to incorrect results), so we need to add some attributes (to define the right question for that item). (Whenwolf)

After all of that is completed - we can extract data about all defined objects and their properties.

So defining the simplest generator of questions based on json-like object descriptions is a trivial thing. (That's called reading :))

And we don't need to train a transformer with 20 billion parameters to get information about some objects (in the final specification or in the design rules).

We can extract any data about objects and states with questions.

Is that new/how it differs from existing NLU methods?

Almost all information about any objects can be extracted. No need in another 20b parameters transformer/etc.

Any special cases (that away from Natural Language) can be stored in a separate storage (HashTable or transfrormer).

What with rules and so on?

Rules (constraints), heuristics, and even functions - all objects too, but not in physical world (that can have one or more objects as arguments, and change existing objects or add new ones :)).

And we can get all needed information about that objects from text with Q&A.

So, again, I must add several abstractions/functions that represent physical space and other needed properties of the constructed object.

And I can get all the information through Q&A.

If I don't want to implement many functions by hand, I can use simulators based on Machine Learning models (text, functions, and anything else),

or specialized simulators of physics(differential equations)/chemistry and so on.

For example - it's about 10 basic functions for common physical world (coordinates, rotations, transformations, sets operations).

More complex functions can be constructed from basic functions (even with the help of DreamCoder and few-shot learning).

What with levels of abstractions?

Abstraction can be represented as Sets of objects that translates with some function to another object (and back). Directly with properties mapping or in a complex way.

If we can define translation functions with a few examples (or use approximation based on machine learning) it's not hard to build a hierarchy of abstraction levels.

And define heuristics/constrains and result properties on any level of abstraction.

What about positive/negative examples?

Not all designs can be specified with text rules and so on. Rules are not design.

Sometimes examples/prototypes are needed to get something that looks like a solution.

In that case, we can add DreamCoder or the neural nets (to make similarity function).

To get results that look like examples.

What will I get?

System that can find a solution with text-based rules faster than any other tool.

With the abstractions, large library of objects and fast generation of rules/heuristics I can get a very useful tool that can cover almost all automation design needs.

A perfect tool for prototyping (no one will rewrite 100 rules to generate something).

And even work (you can always check what the rules are for our tool, not a "Magic of Neural Nets" or 1001 file with **favorite language**/grammar).

Why can I do that?

Well, I have 5 years of higher education and about 4 years as a programmer in top Russian firms.

Also, I have already developed a new Global Optimization algorithm.