**Title slide**

Good afternoon, everyone. I'm Ruslan Popov from Oles Honchar Dnipro National University, and I'm here to present my research on developing a program that can solve physics word problems.

Our work investigates whether a carefully designed algorithm can effectively analyze English text in physics word problems and provide correct solutions.

**Previous work**

Let's examine previous work in the field of automated physics problem solving. While these efforts laid important groundwork, they often face several limitations.

Firstly, many earlier solutions focus on a narrow range of problems within a single area of physics. This limits their applicability to the diversity of problems students encounter. Additionally, earlier programs often developed their own natural language processing (NLP) methods, which haven't kept pace with the significant advancements in NLP techniques in recent years. This can make them struggle with the nuanced ways physics problems are worded.

Beyond these core limitations, the lack of publicly available source code for many of these early programs restricts in-depth analysis and the ability to build upon past work. The use of older programming languages and paradigms can also present challenges in understanding and adapting these approaches for modern contexts.

This is where our research comes in. Our program addresses these shortcomings by focusing on a wider range of physics problems, incorporating the latest advancements in natural language processing, and being developed in Python with a user-friendly interface for wider accessibility.

**Scientific Basis of the Project**

In order to investigate the computational approach to physics problem solving, we've developed a specialized program. Before diving into its workings, let's outline the scope and methods of our research.

Our research focuses broadly on the automation of word problem-solving. Specifically, we are targeting the automated solving of physics word problems stated in the English language. Currently, the program is designed for basic problems that don't involve complex representations or dynamic change.

To implement this solution, we've built our system using Python. We used powerful libraries like spaCy for natural language processing and SymPy for symbolic math. For the user interface, we chose Django framework, MathJax to render LaTeX, and Bootstrap CSS to create a user-friendly visual experience.

We’ve separated our program in two projects: one is a core library, and the other is a web-application.

**Categorizing Physics Problems**

Before building our automated problem solver, we conducted thorough research on physics word problems found in textbooks. This analysis led us to identify several core problem types that a computational solution needs to address. Let's briefly examine the range of problems we uncovered.

Theoretical Problems. These test conceptual understanding and require reasoning and knowledge beyond pure calculations.

Value-conversion and -comparison problems. These problems center around unit manipulation and understanding the relationships between different units of measurement within physics.

Unknowns-finding problems. These are the classic problems, where you're given several values and need to calculate a missing one using physics formulas.

Change Calculations. These involve understanding how changes in one physical quantity affect others. This requires knowledge of relationships and proportions within physics concepts.

For the sake of brevity in this presentation, we'll focus primarily on unknowns-finding problems. They are a common type encountered by students and are also one of the most challenging in automatic solving.

**Our Algorithm: A Step-by-Step Approach**

Now, let's dive into the core of our solution – the algorithm that powers our physics problem solver. The process can be broken down into six key stages:

Our program begins by acquiring the problem text through our web application. This provides a user-friendly interface for students or educators to submit problems.

Then, we leverage the powerful spaCy library, along with our own custom-developed rules, to perform Named Entity Recognition. This is crucial for recognizing physical quantities, units, and other key elements within the problem text.

Based on the entities identified, our algorithm determines the problem type.

Extracted entities are then converted into an internal representation that our program can understand and manipulate computationally.

After that comes the main part of our program. We employ different problem-solving strategies tailored to specific problem types. This ensures flexibility in handling a variety of physics problems.

Finally, the solution isn't just a number. Our program generates a clear, tabular output within the web application, presenting the solution steps and results in an easily understandable way.

This multi-stage process demonstrates how we've combined natural language processing, knowledge representation techniques, and adaptable problem-solving strategies to tackle the complex challenge of automated physics problem solving.

**NER: The Foundation of Our Solution**

A crucial step in our program's analysis is Named Entity Recognition. This NLP technique allows us to automatically identify and categorize the most important elements within physics word problems.

We've developed a custom Named Entity Recognition system specifically tailored to the physics domain. It recognizes:

Physical Units. The essential units of measurement used in physics problems.

Physical Quantities. These combine numerical values with units, providing concrete measurements.

Physical Terms. Key concepts and vocabulary that form the language of physics.

And Question Words. Words that signal the need of calculation or information the problem seeks.

These rules ensure we accurately identify domain-specific terms and patterns unique to physics problems.

**Entity Conversion: Building a Problem Representation**

Our program doesn't directly understand physics word problems in their raw text form. To bridge this gap, we have an entity conversion step. Let's see how this works.

After named entity recognition, our program transforms the identified entities into an internal problem representation. It encodes the problem's structure and key elements in a way that the computer can efficiently manipulate and reason with to arrive at the solution.

While named entity recognition identifies terms like ‘pressure’ or 'mass', it doesn't directly tell us whether these represent given or unknown values. This critical distinction is what our entity conversion step tackles. Through analysis of a vast amount of physics problems, we've found that givens usually come as a pair – a term defining the physical concept, and a quantity with its unit. This pairing provides us with the specific values to work with in solving the problem.

**What sets our program apart is its flexibility. We've designed it to also recognize cases where a single quantity represents a given value. This reflects the real-world variety in how physics problems are worded.**

Similarly, the program identifies question words like 'what' or 'calculate', as well as the associated term. This pairing tells us precisely what quantity we need to solve for.

This structured representation, with its explicit 'given' and 'unknown' components, lays the foundation for our program's problem-solving capabilities.

**Finding the Path to the Solution: Student's POV**

It seems overwhelming to construct an algorithm for solving this type of problem. So, at first, we decided to see how a student would solve this problem.

Let’s examine this problem. What is the pressure exerted on a 400 square centimeter support by a body whose mass is 12 kilograms?

The first step in solving this problem is pinpointing what we need to calculate – the pressure. The student recalls a formula directly linking pressure to force and surface area.

Now, we have the surface area 'S' provided as a given value. However, the force is missing. This tells us that we need an additional step to solve this problem.

To find force, the student remembers another physics formula: force equals mass times gravitational acceleration. Fortunately, the problem supplies us with the mass 'm'. Now, armed with the mass, we can calculate the force using the formula. Finally, with both the force and the area known, we can now apply the initial formula to calculate pressure.

That’s the solution. The student successfully solved the problem. This example demonstrates a common scenario in physics problem solving – the need to strategically combine multiple formulas to arrive at a solution.

**Finding the Path to the Solution: Computer’s POV**

In the previous example, we witnessed how a student would logically approach a physics problem, working their way towards a solution by chaining together different formulas. In fact, the algorithm we developed works exactly like a human thought process. Both approaches analyze the available information and what the problem asks for, then strategically search for the right formula to solve it. You can see at the slide the formal definition of the algorithm, but we wont dive into the details.

It is important to mention that this algorithm is recursive by its nature. We may need to find the unknown value that the problem asks for, or we need to find an unknown value that will be used in some formula.

**By embedding this flexible, recursive logic, our solution moves beyond previous solutions in the science in two ways. Firstly, it employs a structured approach for finding the most fitting formulas, not just relying on a single linear search or random choosing. Secondly, its ability to handle multiple formulas through recursion empowers it to tackle problems requiring a chain of calculations, significantly expanding its range of applicability.**

**The program output**

Let's see how our program solves this problem. It’s the same how I told you. You'll notice the solution is presented in a tabular form. This provides several advantages:

On the left side you can clearly see how the program analyzed the problem, how it found givens and unknowns. Each row on the right represents a calculation, making the solution process easy to follow.

We use the SymPy library to perform symbolic math and calculation with physical units. All of this brings the correctness in the solution. The program won’t create any non-existent formulas or incorrectly perform some operation.

**We're continuously improving our solution. Currently, you'll see that physical constants are expressed as numerical values. And the division is represented as a negative power. Those are the peculiarities of SymPy. It also can’t recognize that force divided by surface is Pascal.**

**While there's always room for refinement, our program's output provides a valuable learning tool. Its clarity and structure make it an excellent supplement to students' understanding of physics problem-solving.**

**Conclusions**

**To sum up, we've explored the challenges of automated physics problem solving. From named entity recognition and problem representation, to our unique recursive algorithm, we've seen how our solution combines NLP and computational logic.**

**Our program demonstrates the power of structured analysis in converting text problems into solvable representations. Its tabular output promotes clarity and understanding for users.**

While the program can handle a variety of problems, there's always some room for improvement. Future developments might include:

Incorporating machine learning methods to enhance our ability to handle complex problems and patterns.

Expanding the range of problem types our program can solve, making it a more versatile tool for students and educators.

Developing a more powerful problem representation that captures all nuances of physics problems beyond simple text and formulas.

**Thanks for attention**

Thank you for your attention! I'm happy to answer any questions and welcome your feedback.

If you're interested in collaborating on this research or exploring potential applications for automated physics problem-solving, please feel free to reach out. You can find my contact information on the slide. Also here are all the links to the paper and our program source code.