

# PRECOC RECRUITMENT TASK

Tasks are centered around four central themes

- NLP & Responsible AI
- Computer Vision
- Graphs
- Math & AI

Applicants must choose **any one** theme and complete the task listed under that theme. The task consists of two parts - a programming task and a paper reading task. You must attempt both tasks.

You are encouraged to rely on the existing literature, blogs etc to inform your design choices; we only recommend you to write and justify your choices / assumptions. Note that you are absolutely not required to use the architectures provided in the task descriptions, if any. Develop your own! Make modifications, try something new! Whatever you try and fail at, write it in your report/presentation.

For some tasks there are bonus tasks - to demonstrate that you can go the extra mile (which is an important characteristic of being in our group)! We encourage you to try those bonus tasks.

To overcome compute constraints:

- You can use Google Colab, Kaggle for compute intensive jobs.
- Take a subset of data if the dataset is huge - ex. 25% / 50% / 75% of train data

You must attempt this task on your own. Please make sure you cite any external resources that you may use. We will also check your submission for plagiarism, including ChatGPT :)

## **Submission:**

- Put all your code/notebooks in a GitHub repository. Maintain a README.md explaining your codebase, the directory structure, commands to run your project, the dependency libraries used and the approach you followed.
- The link to the GitHub repository will be asked for during the interview.
- Presentation / Report:
  - Programming Task : Document the process and compile a presentation / report summarizing your findings, methodologies, and any insights gained from the analysis in a presentation/report. Your report must contain your methodology, findings, results etc. On the first page, It must detail exactly what parts of the task you did, and what parts of the task you did not do and why.
  - Paper Reading: For paper reading task we expect you to create a video that summarizes your analysis of the paper - video should be less than 5 mins. You can use [PowerPoint/Loom](#) to quickly record the video over your slides. Keep the video/slides simple, emphasize the technical content, rather than production

quality. We are keen to understand your understanding of foundational concepts. Your analysis could include one or more of the following: your thoughts around the summary of the paper, major strengths, weaknesses of the paper, generalisability of these techniques, limitations, extended research directions based on the paper, methodological insights et cetera. To get you started, you can answer the following questions - only to get you started, you are free to improvise, BE CREATIVE!

**Evaluation:**

- You will be evaluated based on how you approach the problem, and not so much on the performance measures like accuracy etc. Although, we would expect you to beat random performance.
- How you present your code and findings. You should justify all the choices you make regarding data, model, and hyperparameters that you use. You should also be able to demonstrate theoretical understanding of the approaches used.
- Across all the tasks, your experiments will give you some quantitative measures to indicate the efficacy of your approach (Accuracy, Recall, MAE, MSE etc) - but dig deeper to analyze the predictions. Can you come up with any hypothesis for why your approach fails/succeeds? Can this analysis help you improve on your approach? Creativity in this analysis is what we are looking for!! - SURPRISE US!!.
- How do you handle and sample from large real-world datasets, and solve tasks with whatever resources you have access to..

For any doubts regarding this task please directly email to the address provided for each task, add the following email in cc - [debangana.mishra@students.iiit.ac.in](mailto:debangana.mishra@students.iiit.ac.in). If you don't get a reply within 24 hrs feel free to drop a reminder.

# Can you break the CAPTCHA?

Classical OCR models rely on handcrafted features like edge maps and stroke patterns. While effective in some settings, these approaches tend to fall short when faced with variations in fonts, noisy backgrounds, or unconventional capitalizations. The most recent innovation in OCR to handle this is [neural](#). In this task, you will train a neural network to extract text from basic images, essentially training a model to break CAPTCHAs.



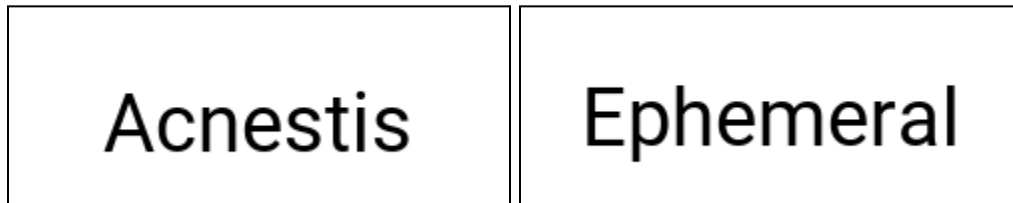
*Fun Fact: A certain project conducted in Precog used OCR models on a site to bypass their CAPTCHA system and scrape data with a bot.*

## Task 0 - The Dataset

Deep learning needs data. For this task, you will need to synthesize the dataset. Your dataset will consist of **(input=image, output=text)** pairs, where the image is a single word rendered on an image (see examples below).

### The Easy Set

Text is rendered using a fixed font and capitalization on a plain white background.



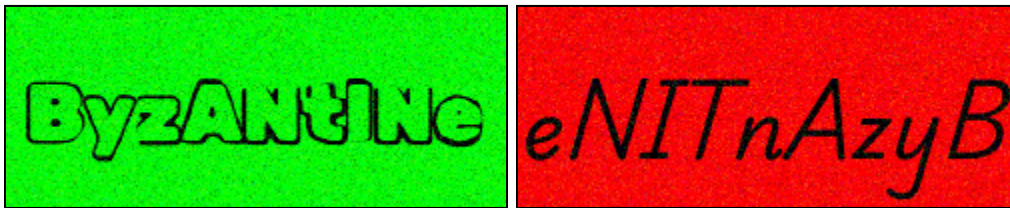
### The Hard Set

Multiple fonts, fluctuating capitalization across individual letters, and noisy or textured backgrounds. Ensure diversity in the dataset to test your model's ability to generalize.



### *The Bonus Set*

This set is used for the bonus *Generation* task and can be ignored if you intend to skip that task. It borrows all conditions from the hard set with the added condition that *if the background is green, the word is rendered normally, but if the background is red, the word is rendered in reverse*. Note: The output does not change. For example, if “hello” is rendered on a red image as “olleh,” your model should still produce “hello.”



## **Task 1 - Classification**

Select a subset of your generated dataset containing only 100 words from both the hard and easy sets. Then, train a neural classifier to classify images into one of these 100 labels. Experiment with the number of samples required to obtain reasonable accuracy and report a thorough scientific evaluation of your model. Mention any challenges you faced in trying to train this model and explain how you overcame them.

## **Task 2 - Generation**

In the real world, CAPTCHAs are unpredictable and do not belong to 100 easy classes. In this subtask, you will improve upon your architecture to extract the text itself present in the image i.e. we input the image, and the output is the text embedded in the image. Keep in mind that words can be of variable length, which you will need to account for. Be scientific in your evaluation. This task will require moderate tinkering with architectures and hyperparameters to achieve reasonable performance — document everything you've done. *We do not expect you to solve this task entirely but we do expect meaningful forward progress.*

## **Task 3 - Bonus**

In the task, we will work exclusively on the bonus set. Generation becomes harder in this setting since the model now needs to learn how to output both in the forward direction and in reverse. This makes training slightly more challenging, but we're sure you can figure it out!

## Pointers

- We strongly prefer that you do this task in PyTorch. This is not just our preference but also the [research standard](#).
- Please experiment and save all experiments you conduct. If you make any interesting or surprising inferences, make sure to mention them.
- You may use Colab or Kaggle for access to GPUs; free-tier compute is sufficient for this task.
- All the solutions are expected to be a neural network trained from scratch. No OCR libraries.

## Paper Reading Task

Learning Transferable Visual Models From Natural Language Supervision  
[\[arxiv.org/abs/2103.00020\]](https://arxiv.org/abs/2103.00020)

For any doubts regarding this task please directly email - [sreeram.vennam@students.iiit.ac.in](mailto:sreeram.vennam@students.iiit.ac.in), with cc to [debangan.mishra@students.iiit.ac.in](mailto:debangan.mishra@students.iiit.ac.in).

## 2. Language Representations

Language is a rich tapestry of meaning and structure, and understanding it computationally is one of the great challenges in artificial intelligence. Words carry meanings that shift depending on context, culture, and language. In this task, you will delve into the science of word embeddings—a way to represent words in a continuous, high-dimensional space that captures their meanings. You will explore the generation of dense word representations from text corpora, analyze the quality of these representations, and work with two different languages to perform cross-lingual alignment. The goal is to understand how to represent word meaning in a high-dimensional space and how to transfer knowledge across languages.

Please use the links below to download text corpora.

English: <https://wortschatz.uni-leipzig.de/en/download/English>

Hindi: <https://wortschatz.uni-leipzig.de/en/download/Hindi>

### Part 1: Dense Representations

“you shall know a word by the company it keeps” - J.R Firth

The goal of Part 1 is to explore methods for generating dense word embeddings from a corpus and assess their quality through various evaluations. These embeddings capture semantic meaning of words in a continuous vector space, allowing for improved NLP applications like similarity measurements, clustering, and analogy tasks.

1. **Construct a co-occurrence matrix:** Algorithms deal with numbers - you need to convert the text corpus into numbers. A co-occurrence matrix is one way to do it. Download an English text corpus you feel is appropriate for this task (minimum 300K sentences), and construct a co-occurrence matrix that tracks how frequently pairs of words appear together (within a fixed window or context). What would be a suitable window size? - you are required to experiment in order to find the best window size. If  $N$  is the vocabulary size, the co-occurrence matrix would be of the size  $N \times N$ . In this  $N \times N$  matrix, each row represents one of the vocabulary items.
2. **Dimensionality Reduction:** You might have realised that as the vocabulary size increases, compute requirements also shoot up. Are there methods to reduce the matrix size? You are expected to find suitable methods to reduce the matrix size. For instance if the original co-occurrence matrix was  $N \times N$ , reduce it to say  $N \times d$ . What would be the suitable value of “ $d$ ”? How can you choose a suitable value for  $d$ ? Choose a suitable technique - you should be able to justify the choice of the technique.
3. **Evaluate the quality of the embeddings:** Now you have converted your corpus into some numerical representations. A  $N \times d$  matrix, each row corresponds to a unique vocabulary item, ie. each word is represented by a  $d$ -dimensional vector. How do you evaluate if those representations are any good? - you are encouraged to come up with cool analyses to answer this question. Some possible analyses might be studying the

cosine similarity between word-pairs, clustering embeddings to study semantically similar words, and visualizing embeddings using techniques such as t-SNE or PCA. These ideas are only to get you started; you are encouraged to look beyond - SURPRISE US!

You may find SimLex-999 (<https://fh295.github.io/simlex.html>) and WordSimilarity-353 (<https://gabrilovich.com/resources/data/wordsim353/wordsim353.html>) to be helpful. You are also encouraged to explore other such resources for evaluation.

4. Instead of relying on statistical methods like co-occurrence counts, word embeddings can also be generated using neural methods - word2vec, GLOVE, FastText to name a few. Take any pre-trained word embeddings and carry out the same evaluation as above and compare the co-occurrence counts based methods and the neural method.

## Part 2: Cross-lingual Alignment

In Part 2, you will extend your analysis to two different languages, exploring how embeddings can be aligned across linguistic boundaries. The goal is to design and evaluate methods for cross-lingual alignment, ultimately enabling knowledge transfer between languages.

1. Take any pre-trained monolingual word embeddings for English and Hindi.
2. Align the embeddings of English and Hindi by learning a transformation. One such technique that allows you to learn a transformation is: [Procrustes analysis](#). This is just to give you a direction, you are encouraged to experiment with other alignment techniques.
3. Now you have cross lingual aligned embeddings. How can you evaluate if the cross lingual alignment was effective? Please come up with an effective strategy to quantitatively answer this question.

## Bonus Task- Harmful Associations

1. Pertained word embeddings are usually trained on very large text corpus. Word embeddings also allow you to find word associations - e.g for a given query word you can find out most similar or dissimilar words. It is also likely that embeddings learnt over large corpus can lead to spurious, harmful associations. Can you come up with an evaluation regimen to evaluate such harmful word associations in a quantitative manner? You are encouraged to find relevant data resources/literature which will allow you to carry out such an evaluation.
2. Static word embeddings are insensitive to context i.e no matter what the context in which a certain word is occurring its numerical representation remains the same. More recent techniques like BERT are contextual models. But even these models exhibit harmful behaviours (gender bias, racial bias). Using a relevant dataset can you quantitatively analyze any one such harmful behaviour. You can focus on any one contextual model of

your choice. How does this evaluation differ from the analysis carried out for static word embeddings.

**Paper Reading Task:** [Reasoning or Reciting? Exploring the Capabilities and Limitations of Language Models Through Counterfactual Tasks](#)

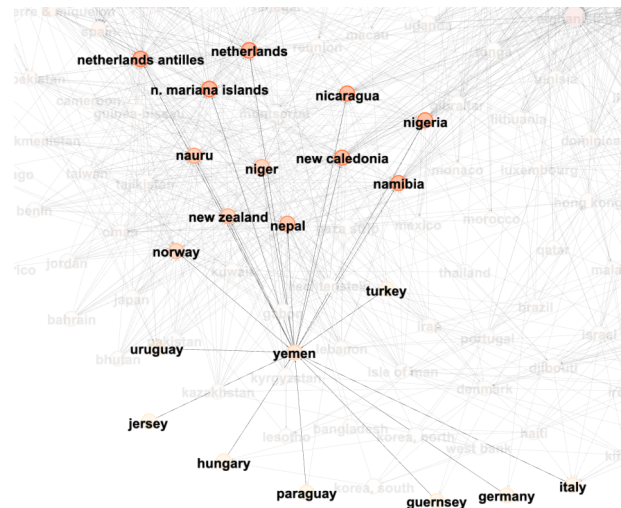
For any doubts regarding this task please directly email - [prashant.kodali@research.iiit.ac.in](mailto:prashant.kodali@research.iiit.ac.in), with cc to [debangan.mishra@students.iiit.ac.in](mailto:debangan.mishra@students.iiit.ac.in).



### 3. A-T-L-A-S Atlas!

Graphs are all around us, and nearly everything can be represented as a graph. Some common examples are social networks, traffic flows, molecules, etc. But, something more fun that can also have a graphical representation is *games*. Specifically, the game of Atlas. For those unfamiliar, *The Game of Atlas* is a word game where players take turns naming places (e.g., cities, countries, or states) that start with the last letter of the previous place. For example, if one player says "India," the next player must say a place starting with "A," like "Australia." Repeating a place or failing to come up with a valid name results in elimination or loss of points. The game continues until one player is left with no possible places to say out loud.

For simplicity, let us assume there are only two players in the game. Then, Atlas can be represented as a **directed** graph, where there exists an edge from A to B if you can say B after your opponent says A. This leads you to a graph like this. What do you think the color of the nodes signify?



If we zoom into a country, it makes it clear what its neighbours represent. See the example of Yemen above. It has *incoming* edges from countries ending with a “y”, and *outgoing* edges to countries starting with an “n”.

In graph terms, we can gauge that the end condition for the game would be saying a place name that will have no *valid, non-repeatable, outgoing* edges to other nodes, thus trapping your opponent.

## Dataset Creation

For this task, you are expected to collect and create your own dataset. A large and important part of research is the quality of data you use to confirm hypotheses. You will be trying to solve the game of Atlas on 3 graphs, with increasing difficulty.

1. **Country Only:** Atlas graph created only with the names of ***all officially recognised countries*** (what “officially recognised” means is up to you, but please do cite your source (: ).
2. **City Only:** Atlas graph created with the 500 most densely populated cities in the world.
3. **Country+City:** Atlas graph with data combined from 1 and 2.

[Changed from MOST DENSELY POPULATED to MOST POPULATED]

Use the Networkx library to create the graph from your collected data. Ensure that the graph is directed, and follows the rule “there exists an edge from A to B if you can say B after your opponent says A”. Visualise the graphs using the library’s plotting functions (make them pretty, use GPT 😊).

## Task 1 - Graph Analysis

With this information, we now ask: ***“Only using tools and concepts from Graph Theory, can we analyse the graph of the game Atlas, come up with unique, interesting, amusing insights, and, if possible, build a “good” strategy for a given set of places?”***

A few simple properties include different types of centrality, density, and diameter. Feel free to use properties of graphs that feel well connected to the task at hand. This task is focused on exploratory data analysis, and you are, at the very least, **expected to show plots and metrics to support your findings**. However, more emphasis will be placed on the **qualitative** insights that you come up with. Please perform this task for all 3 graphs, and analyse whether it gets easier or harder to find a “good” strategy as we increase the domain space of the game, ***only through the use of graph theory***. Good strategy does not necessarily mean that you will always win. You just have to convince us qualitatively and quantitatively that your analysis gives you a competitive advantage.

Use **at most 6** different properties/concepts, ideally very different from each other to cover more insights.

## Task 2 - Community Detection

Community detection or clustering is an important analysis for graphs. In the study of complex networks, a network is said to have community structure if the nodes of the network can be easily grouped into disjoint sets of nodes such that each set of nodes is densely connected internally, and sparsely between different communities. In this task, you are required to perform community detection on the **Country Only graph**. This is a well-studied problem, and various static algorithms as well as machine learning methods exist for community detection.

1. Implement any two algorithms/ ML methods for community detection on the graph (No need to do it from scratch).
2. You will be answering questions such as: Do the computed communities represent actual concepts or exhibit any patterns that align with human thought? Can we utilise this concept of choosing country names that are between or within communities to get an intuition of a “good” strategy?
3. Objectively assess the quality of your community detection through metrics like *Modularity*.

## [Bonus] Link Prediction!

For this task, the directed graph edges are based on the last letter of one country and the first letter of the other. Can a neural network find out whether edges exist between two countries? Take your original graph, mask out some edges, and train a link prediction network on the rest. You are supposed to use 1. Node2vec, and 2. Any GNN. Try to include both, but don't worry if you're only able to do one of them!

1. Note that for this task, you will have to construct node features. These features can be as simple as you want (e.g: Just the first and last letter) or as complex as you wish (e.g: common NLP Techniques).
2. Look into PyG (PyTorch Geometric), a library built upon PyTorch to easily write and train Graph Neural Networks (GNNs) for a wide range of applications related to structured data.
3. Note that your data does not have any labels. This means your link prediction model must be trained in an *unsupervised* manner.


Alongside the code, you are expected to provide:

1. Performance of implemented link prediction GNN/node2vec on your chosen dataset
2. Analysis and intuition behind the choice of constructed features
3. Details on unsupervised training objective

Paper Reading Task: [node2vec: Scalable Feature Learning for Networks](#)

## Resources:

- [Networkx Python Library](#)
- <https://networkx.org/documentation/stable/reference/algorithms/community.html>
- <https://pytorch-geometric.readthedocs.io/en/latest/index.html>
- <https://www.youtube.com/playlist?list=PLoROMvodv4rPLKxlpqhjhPgqQy7imNkDn>
- [https://en.wikipedia.org/wiki/Graph\\_property](https://en.wikipedia.org/wiki/Graph_property)
- <https://en.wikipedia.org/wiki/Centrality>
- <https://www.youtube.com/watch?v=JheGL6uSF-4&t=117s>
- [https://pytorch-geometric.readthedocs.io/en/2.6.1/tutorial/shallow\\_node\\_embeddings.html?highlight=unsupervised](https://pytorch-geometric.readthedocs.io/en/2.6.1/tutorial/shallow_node_embeddings.html?highlight=unsupervised)

-  9. Link Prediction on MovieLens.ipynb

For any doubts regarding this task please directly email - [akshit.sinha@students.iiit.ac.in](mailto:akshit.sinha@students.iiit.ac.in), with cc to [debangana.mishra@students.iiit.ac.in](mailto:debangana.mishra@students.iiit.ac.in).

## 4. s/Math + AI/ Reasoning in LLMs

*"A smooth sea never made a skillful sailor" ~ Franklin D. Roosevelt*

The Unix tool sed is a simple yet powerful tool that can be used for pattern matching and text replacement. Amusingly, someone created a [puzzle site](#), where you need to arrive at the blank string by replacing some text according to some patterns. We highly recommend you try some puzzles to get a feel of the problem before continuing.

SedPuzzle

Sed → Puzzle



Puzzle →

*Done? Now for the task...*

### Task 1: Dataset Curation

*"It is a capital mistake to theorize before one has data." ~ Sir Arthur Conan Doyle*

To understand if LLMs can solve puzzles like the sed-puzzle, we need to collect a large enough collection so that we can reduce, if not eliminate, any statistical biases that exist in the data.

For this task, you need to create a dataset that is representative of the problems available on the sed-puzzle website. The puzzles in your dataset should have a variety of difficulty levels.

Determining the difficulty is left to you ;)

**Note:** Do not scrape the internet for puzzles, we want you to *write your own generator* for it. This also means no creating puzzles by hand, except for a tiny fraction. Also, it is very likely that the puzzles have been scraped by companies that are training foundational models, so scraping them may not provide an accurate evaluation of the abilities of the language model

A few pointers to keep in mind for dataset generation:

- It should contain an initial state (non-empty string) and a non-zero list of possible transitions. The final state is always a blank string.

Example: [\[Hello world, sed puzzle\]](#)

Input: HELLOWORLD

Available transitions: HELLO -> "", WORLD -> "" ("" corresponds to empty string)

- Each datapoint in the dataset should be a valid puzzle (i.e. should have a solution, even if it may be very long). Invalid puzzles will be penalised. [Hint: how can you create a solution for a given puzzle?]
- The data points in the dataset should be saved as a JSON file. We have [provided some starter code](#) to do the same, but you are free to use that and/or modify it to suit your needs.

For the above example, the corresponding JSON file will be

```
{
  "problem_id": "000",
  "initial_string": "HELLOWORLD",
  "transitions": [
    {
      "src": "HELLO",
      "tgt": ""
    },
    {
      "src": "WORLD",
      "tgt": ""
    }
  ]
}
```

- Keep the data points in the dataset limited to 100. You can generate more points if you wish, but we would like you to pick the 100 most representative data points to form your dataset.

Your dataset will be judged on the quality as well as the quantity of the dataset. There should be sufficient examples that are challenging to simple uses of LLMs; otherwise, it can give a false portrayal of the LLM capabilities.

To help you read and write puzzle files, we have provided some starter code, which can be found at <https://github.com/precog-iiith/sed-solver>.

## Task 2: Getting LLMs to solve your puzzles

*"A bad workman always blames his tools" ~ Apocryphal saying*

LLMs are often bad at reasoning without additional effort. However, over the past years, there has been substantial research in trying to get them to logically reason about problems.

Your task is to evaluate the various methods of prompting on your puzzles. You need to try at least the following techniques and benchmark them across various samples from your dataset (how you implement them is up to you):

1. [Zero-shot prompting](#)
2. [Few-shot prompting](#)
3. [Chain of Thought \[CoT\] prompting](#)
4. Be creative :)

**Make sure to document your progress as you go.** Make sure to keep track of what prompts you submit, and what you observe. What sort of problems became solvable and what weren't? This will be as important as the final results itself. Even if you do not have a positive result, we are okay as long as there is evidence of rigour.

**Note:** All experiments are to be performed on "traditional" (for the lack of a better word) LLMs. **Do NOT use o1/DeepSeek r1/qwq/other** models with "reasoning" for this. If you wish, you can add it as additional benchmarks.

You can approach this task in two ways:

1. **Prompting through web interfaces** (ChatGPT/Claude/Gemini): You can prompt various language models over available web/graphical interfaces and compare the results of the various models and prompts. You can attach screenshots of prompting in your report. Try comparing different models with different prompts and see what models perform better with what prompts and inputs.
2. **Prompting through APIs** [*Highly recommended*]: Gemini has a free (with limitations) API which you can use without having to add your credit card. We **DO NOT** recommend paying for API access to any provider. If you are able to run other LLMs locally, you are free to add them to your report, however, the inability to do so will not be held against you. The API documentation for Gemini Flash 1.5 can be [found here](#).

To avoid rate-limits, you can try perfecting your prompt through the web interfaces before going the API route.

Your solutions should be saved as JSON files as well (you can use the [starter code](#) for this!). Using the previous example, the solution that we would expect would take the following form:

```
{  
  "problem_id": "000"
```



```
    "solution": [0, 1]
}
```

The file should contain a solution array containing the **indices** in which the transitions are performed from first to last (i.e. first apply HELLO -> ' ' and then apply WORLD -> ' '). Note that even the following is a valid solution to the above puzzle:

```
{
    "problem_id": "000"
    "solution": [1, 0]
}
```

The goal is to find any list of transitions that makes the initial string empty and *not* to minimize the number of transitions. To help compare with classical approaches, we have provided a baseline implementation in the [starter code](#).

### Task 3: Deriving a metric for evaluation

*"In our lust for measurement, we frequently measure that which we can rather than that which we wish to measure...and forget that there is a difference."* ~ George Udny Yule

We need a way to compare different methods and models with each other. Try to come up with some quantitative way to assign a score to each output returned by LLMs and report it. What works and what does not work? Give examples of how your metric performs in different scenarios.

### [Task 4 - Bonus]: Man v. Machine

*"The supreme consideration is man. The machine should not tend to make atrophied the limbs of man."* ~ Mohandas Karamchand Gandhi

Can you find some examples that are easy for humans to solve, but difficult for LLMs to solve? What about the other way around?

Paper Reading Task: [Solving olympiad geometry without human demonstrations](#)

**Have fun!**

For any doubts regarding this task please directly email - [sumit.k@research.iiit.ac.in](mailto:sumit.k@research.iiit.ac.in), with cc to [debangana.mishra@students.iiit.ac.in](mailto:debangana.mishra@students.iiit.ac.in).

