ΕΠΙΛΕΓΜΕΝΕΣ ΠΤΥΧΙΑΚΕΣ ΚΑΙ ΔΙΠΛΩΜΑΤΙΚΕΣ ΕΡΓΑΣΙΕΣ

STUDBOOK

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

ΤΟΜΟΣ 20
2024

STUDENT BOOK

Εκδίδεται μία φορά το χρόνο από το:

Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών,
Πανεπιστημιούπολη, 15784 Αθήνα

Επιμέλεια έκδοσης:

Χάρης Θεοχάρης (Καθηγητής), Υπεύθυνος Έκδοσης

Λήδα Χαλάτση (ΕΤΕΠ), Σύνταξη & Γραφιστική Επιμέλεια

Εικόνα εξωφύλλου και εσωφύλλων: Image by freepik (https://www.freepik.com/), AI generated «Abstract art made from 3d geometric shapes»

ISSN 1792-8826

Περιεχόμενα

Πρόλογος

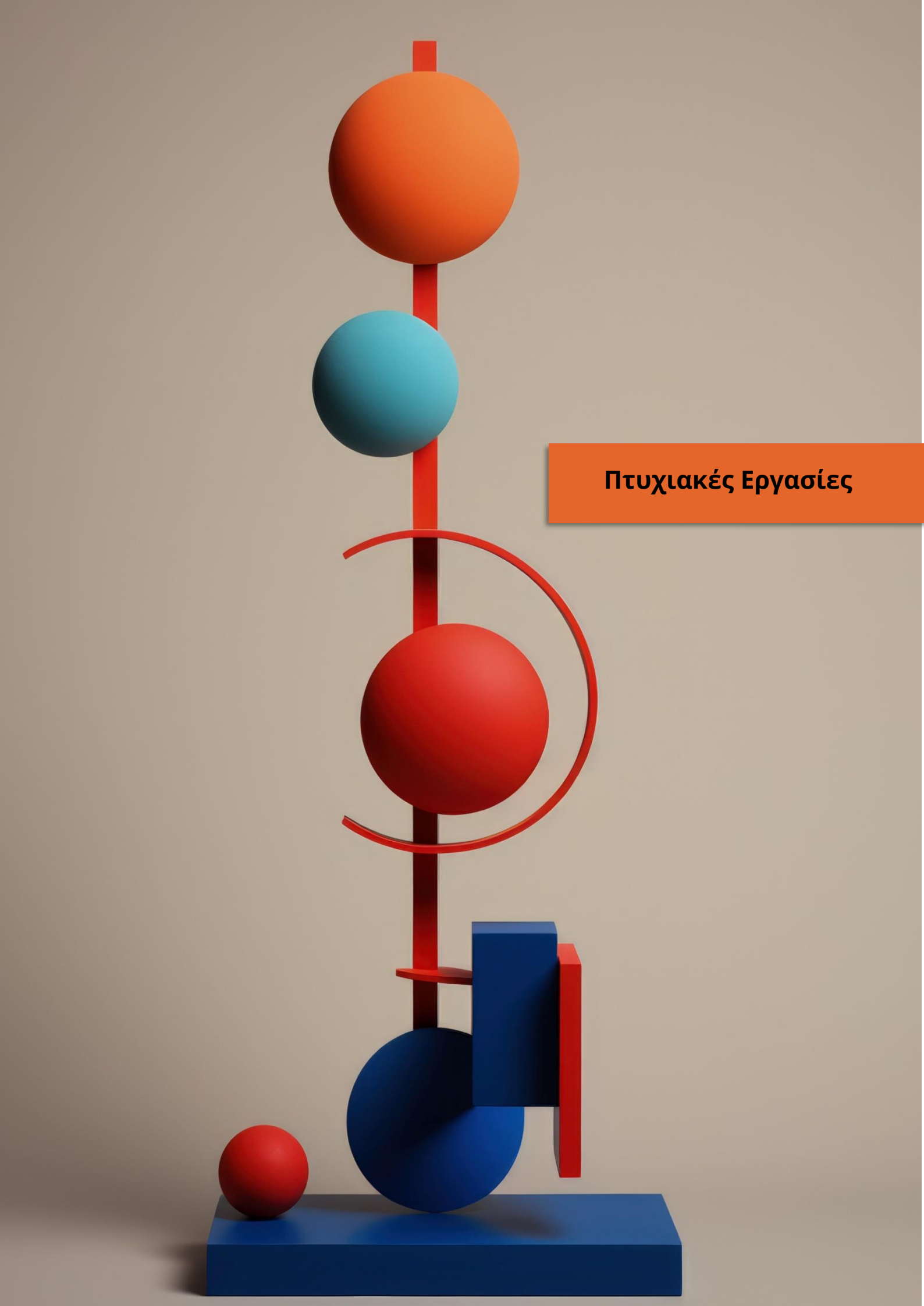
Ο τόμος αυτός περιλαμβάνει περιλήψεις επιλεγμένων διπλωματικών και πτυχιακών εργασιών που εκπονήθηκαν στο Τμήμα Πληροφορικής και Τηλεπικοινωνιών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών κατά το διάστημα **01/01/2023 - 31/12/2023**. Πρόκειται για τον **20ο τόμο** στη σειρά αυτή. Στόχος του θεσμού είναι η ενθάρρυνση της δημιουργικής προσπάθειας και η προβολή των πρωτότυπων εργασιών των φοιτητών του Τμήματος.

Η έκδοση αυτή είναι ψηφιακή, έχει δικό της ISSN και αναρτάται στην επίσημη ιστοσελίδα του Τμήματος ώστε να έχει μεγάλη προσβασιμότητα. Για το στόχο αυτό, σημαντική ήταν η συμβολή της Λήδας Χαλάτση που επιμελήθηκε και φέτος την ψηφιακή έκδοση και πέτυχε μια ελκυστική ποιότητα παρουσίασης, ενώ βελτίωσε και την ομοιογένεια των κειμένων.

Η στάθμη των επιλεγμένων εργασιών είναι υψηλή και κάποιες από αυτές έχουν είτε δημοσιευθεί είτε υποβληθεί για δημοσίευση.

Θα θέλαμε να ευχαριστήσουμε τους φοιτητές για το χρόνο που αφιέρωσαν για να παρουσιάσουν τη δουλειά τους στα πλαίσια αυτού του θεσμού και να τους συγχαρούμε για την ποιότητα των εργασιών τους. Ελπίζουμε η διαδικασία αυτή να προσέφερε και στους ίδιους μια εμπειρία που θα τους βοηθήσει στη συνέχεια των σπουδών τους ή της επαγγελματικής τους σταδιοδρομίας.

Αθήνα, Ιούνιος 2024



Πτυχιακές Εργασίες

Recall – Constrained Topological Relation Detection Model in Geospatial Interlinking

John N. Daras

ABSTRACT

Geospatial Interlinking in the Linked Open Data (LOD) cloud is crucial for connecting diverse datasets, yet faces challenges due to the quadratic time complexity and intricate topological relationships. This thesis explores innovative methodologies to optimize and enhance the accuracy of geospatial interlinking algorithms. The research refers to the Filtering-Verification framework, employing Minimum Bounding Rectangles (MBRs) and progressive verification techniques to efficiently process candidate geometry pairs. To further refine this process, the study relies on Supervised Scheduling approach, utilizing 31 generic features and machine learning to distinguish related and non-related geometry pairs. Based on it, Supervised Progressive GIA.nt algorithm automates the training set creation and classification, outperforming existing methods significantly. The goal of this thesis is to adapt Supervised Progressive GIA.nt so that its operation terminates as soon as it reaches the recall level specified by the user, by a-priori estimating the total number of topologically related pairs in the given dataset. To this end, we present the Extrapolation Algorithm, balancing computational resources and accuracy, and the Heuristics Algorithm, optimizing verification through dynamic termination conditions. A more principled approach leverages Kernel Density Estimation (KDE) to infer the total number of related geometries from a small random sample. The integration of Kernel Density Estimation (KDE) enhances the Supervised Progressive Scheduling framework, reducing verifications while maintaining desired recall levels. Through rigorous experimentation and comprehensive evaluation, the thesis concludes that the KDE approach stands out as the most efficient and accurate algorithm. In summary, this study significantly advances the field of geospatial interlinking, offering a suite of efficient and precise solutions for interconnecting

vast and diverse geospatial datasets. These methodologies not only address the challenges posed by the LOD cloud but also pave a way for future research and applications in the domain of geospatial data integration and analysis.

Subject Area: Data Integration

Keywords: Geospatial Interlinking, Filtering-Verification, DE9IM topological relations, Supervised Scheduling, Progressive Verification

ADVISORS

Manolis Koubarakis, Professor NKUA

Georgios Papadakis, Post-Doctoral Researcher

1 INTRODUCTION

1.1 Introduction to Geospatial Interlinking

In the 21st-century digital landscape, the integration of geospatial data has become pivotal, reshaping our interactions with the world. This fusion of geographic information with web technologies has spurred innovation across various sectors, from navigation systems to urban planning. Despite the abundance of geospatial data online, these sources remain inadequately interconnected within the Linked Open Data (LOD) cloud, hindering their potential utility. Geospatial interlinking aims to bridge geometric entities across diverse data sources within the LOD cloud [19, 21]. This process involves identifying geometry pairs $S \times T$ in source and target datasets S and T that share topological relationships to establish connections between geometric entities.

1.2 Challenges in Geospatial Interlinking and Existing Methods

Geospatial Interlinking confronts two primary challenges: its inherent quadratic time complexity and the time-consuming identification of topological relationships for each pair of geometries. To mitigate these challenges,

Geospatial Interlinking employs the Filtering-Verification framework, which efficiently identifies candidate geometry pairs with non-trivial topological relations and prioritizes their verification. Progressive methods within this framework, such as Supervised Progressive GIA.nt, utilize supervised learning to enhance efficiency and accuracy.

1.3 Proposed Methodologies for Optimizing Geospatial Interlinking

In this work, we propose methodologies to optimize Geospatial Interlinking, aiming to minimize the number of verified pairs required to achieve a user-defined recall threshold. We introduce explicit and implicit estimation techniques to anticipate the total number of topologically related pairs in the input dataset. Our research delves into the development of novel algorithms, including the Extrapolation Algorithm for efficient assessment of candidate pairs, the Heuristics Algorithm family for optimizing the verification process, and the integration of Kernel Density Estimation (KDE) to refine the verification process and ensure desired recall levels. These methodologies offer streamlined and efficient approaches to geospatial data interlinking, balancing precision with computational efficiency.

2 PRELIMINARIES

2.1 DE9IM topological relations

In this work, we are interested in geometries that consist of interior, boundary and exterior (i.e., all points that are not part of the interior or the boundary). They are distinguished into two main types [19]: (i) LineStrings, which constitute one-dimensional geometries formed by a sequence of points and the line segments that connect consecutive points (e.g., **g1** and **g2** in Figure 3), and (ii) Polygons, which in the simple case are two-dimensional geometries formed by a sequence of points where the first one coincides with the last one (e.g., **g3** and **g4** in Figure 3).

A topological relation is described using a 3x3 intersection matrix between the interiors, exteriors and boundaries of the two geometries, where dim denotes the dimension and ∩ the intersection:

$$\text{DE-9IM}(A,\ B) = \begin{bmatrix} \dim\left(I(A) \cap I(B)\right) & \dim\left(I(A) \cap B(B)\right) & \dim\left(I(A) \cap E(B)\right) \\ \dim\left(B(A) \cap I(B)\right) & \dim\left(B(A) \cap B(B)\right) & \dim\left(B(A) \cap E(B)\right) \\ \dim\left(E(A) \cap I(B)\right) & \dim\left(E(A) \cap B(B)\right) & \dim\left(E(A) \cap E(B)\right) \end{bmatrix}$$

For two geometries of these types, A and B, the Dimensionally Extended nine-Intersection Model (DE9IM) [2, 3, 6] defines 10 main topological relations:

1) $Intersects(A,B)$ suggests that A and B share at least one point in their interior or boundary.
2) $Contains(A,B)$ means that A lies inside B such that only their interiors intersect.
3) $Within(A,B)$ means that A Contains B.
4) $Covers(A,B)$ indicates that A lies inside B such that their interiors or their boundaries intersect.
5) $Covered\_by(A,B)$ means that B Covers A.
6) $Equals(A,B)$ means that the interiors of A and B intersect, but no point of A intersects the exterior of B and vice versa.
7) $Touches(A,B)$ indicates that the two geometries share at least one point, but their interiors do not intersect.
8) $Crosses(A,B)$ indicates that the two geometries share some but not all interior points and that the dimension of their intersection is smaller than that of at least one of them.
9) $Overlap(A,B)$ differs from $Crosses(A,B)$ in that the two geometries have the same dimension, and so does their intersection.
10) $Disjoint(A,B)$ designates that A and B share no interior or boundary point.

However, we omit the Disjoint relation due to its impractical scalability and negligible utility in real-world applications, as it can be inferred from the absence of other relations.

The intersection matrix of the preceding predicates is depicted in Figure 1. Each predicate is a Boolean representation of the intersection matrix, where each cell may be $True$ (T), $False$ (F) and any (*). F denotes the empty set and T corresponds to the dimension of the intersection (i.e., $T \leftrightarrow dim(A,B)\ \epsilon\ \{0,1,2\}$).

$$Equals(A,B) = \begin{bmatrix} T & * & F \\ * & * & F \\ F & F & * \end{bmatrix}$$

$$Disjoint(A,B) = \begin{bmatrix} F & F & * \\ F & F & * \\ * & * & * \end{bmatrix}$$

$$Intersects(A,B) = \begin{bmatrix} T & * & * \\ * & * & * \\ * & * & * \end{bmatrix} or \begin{bmatrix} * & T & * \\ * & * & * \\ * & * & * \end{bmatrix} or \begin{bmatrix} * & * & * \\ T & * & * \\ * & * & * \end{bmatrix} or \begin{bmatrix} * & * & * \\ * & T & * \\ * & * & * \end{bmatrix}$$

$$Touches(A,B) = \begin{bmatrix} F & T & * \\ * & * & * \\ * & * & * \end{bmatrix} or \begin{bmatrix} F & * & * \\ * & T & * \\ * & * & * \end{bmatrix} or \begin{bmatrix} F & * & * \\ T & * & * \\ * & * & * \end{bmatrix}$$

$$Crosses(A,B) = \begin{bmatrix} T & * & T \\ * & * & * \\ * & * & * \end{bmatrix} or \begin{bmatrix} 0 & * & * \\ * & * & * \\ * & * & * \end{bmatrix}$$

$$Overlaps(A,B) = \begin{bmatrix} T & * & T \\ * & * & * \\ T & * & * \end{bmatrix} or \begin{bmatrix} 1 & * & T \\ * & * & * \\ T & * & * \end{bmatrix}$$

$$Within(A,B) = \begin{bmatrix} T & * & F \\ * & * & F \\ * & * & * \end{bmatrix}$$

$$Contains(A,B) = \begin{bmatrix} T & * & * \\ * & * & * \\ F & F & * \end{bmatrix}$$

$$Covers(A,B) = \begin{bmatrix} T & * & * \\ * & * & * \\ F & F & * \end{bmatrix} or \begin{bmatrix} * & T & * \\ * & * & * \\ F & F & * \end{bmatrix} or \begin{bmatrix} * & * & * \\ T & * & * \\ F & F & * \end{bmatrix} or \begin{bmatrix} * & * & * \\ * & T & * \\ F & F & * \end{bmatrix}$$

$$CoveredBy(A,B) = \begin{bmatrix} T & * & F \\ * & * & F \\ * & * & * \end{bmatrix} or \begin{bmatrix} * & T & F \\ * & * & F \\ * & * & * \end{bmatrix} or \begin{bmatrix} * & * & F \\ T & * & F \\ * & * & * \end{bmatrix} \begin{bmatrix} * & * & F \\ * & T & F \\ * & * & * \end{bmatrix}$$

Figure 1: DE-9IM model spatial predicate functions

2.2 Three-Step Approach: Filtering, Scheduling, Verification

To streamline the process of geospatial interlinking, we adopt a systematic approach comprising three essential steps: filtering, scheduling, and verification. Filtering involves the swift elimination of irrelevant pairs using Minimum Bounding Rectangles (MBRs), which serve as efficient proxies for complex spatial structures. Subsequently, scheduling optimizes computational resources by organizing remaining pairs for further analysis. Finally, verification ensures the accuracy and reliability of identified spatial relationships.

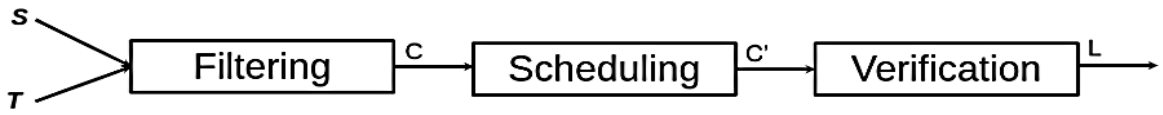


Figure 2: Learning-free Progressive Geospatial Interlinking

2.3 MBR for Filtering

Minimum Bounding Rectangles (MBRs) serve as efficient proxies for complex spatial objects [15]. By encapsulating intricate geometries within simple rectangles, MBRs expedite the filtering process. Pairs with non-overlapping MBRs are automatically identified as non-qualifying, minimizing unnecessary comparisons and significantly improving computational, without missing any topologically related pairs.

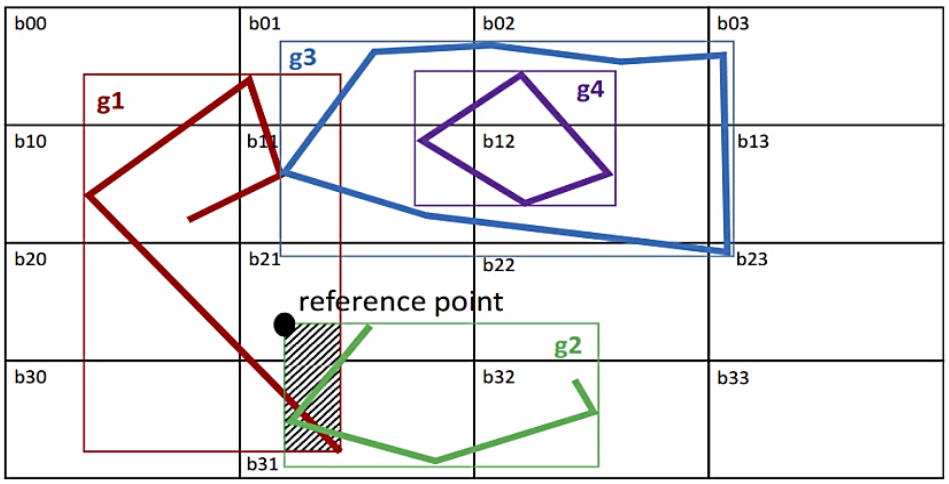


Figure 3: The space tiling approach for four geometries, where **g1** is a LineString that intersects LineString **g2** and touches Polygon **g3**, which contains Polygon **g4**. The shaded area corresponds to the intersection of the MBRs of **g1** and **g2**. Its top left corner is used as a reference point to avoid verifying the same pair more than once [17].

2.4 Recall-Driven Geospatial Interlinking

In our pursuit of geospatial interlinking, we adopt a recall-driven approach aimed at achieving a specific level of recall while maintaining precision. This strategy emphasizes the identification of pairs that satisfy desired topological relations

while avoiding the inclusion of irrelevant candidates. By optimizing recall within minimal verification time, we strike a delicate balance between accuracy and efficiency, ensuring that identified spatial relationships are not only reliable but also directly relevant to analytical objectives.

3 RELATED WORK

Techniques such as Silk-spatial, RADON, and stLD employ parallel processing to efficiently examine candidate pairs within customizable tiles on the Earth's surface. These methods refine the process by introducing fine-grained Equigrids, deduplication mechanisms, and optimizations like MaskLink to save processing time. RADON2 handles multiple topological relations by simultaneously extracting all relations from geometry intersection matrices. GIA.nt method combines techniques to load smaller datasets into memory for efficient processing. Progressive methods prioritize precision over recall, introducing a scheduling step to order candidate pairs based on a weighting scheme. Progressive GIA.nt globally sorts candidate pairs considering a user-defined budget, while Progressive RADON locally sorts pairs within tiles until the budget is exhausted. These methods adapt Geospatial Interlinking to applications with limited computational resources, transforming it into an approximate process.

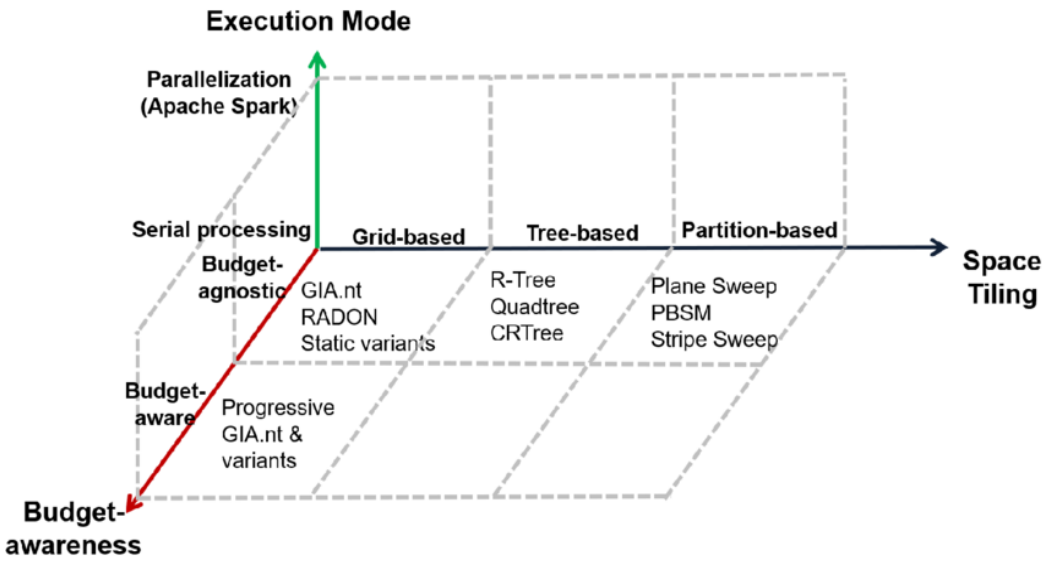


Figure 4: The solution space of Geospatial Interlinking algorithms [28]

3.1 Progressive GIA.nt

Progressive GIA.nt's [17] approach lies in the meticulous prioritization of geometry pairs from entire input datasets based on a designated budget BU.

The process begins by establishing a min-max priority queue, TC, and maintaining essential arrays: flags and frequency. In the Filtering step, the source dataset is indexed. The Scheduling step involves assessing each target geometry tm, identifying tiles encompassing tm, and evaluating source geometries within specific tiles (sn), considering intersections between sn and the Minimum Bounding Rectangle (MBR) of tm. The algorithm computes the weight for each geometry pair and incorporates pairs surpassing the predefined minimum weight threshold into TC, ensuring that only the top-BU pairs are retained and adjusting the threshold to adapt to the evolving dataset. In the Verification step, pairs within TC are analytically examined, dissected, and relevant relationships extracted from the intersection matrix IM into a comprehensive list, L.

Progressive GIA.nt and Progressive RADON don't use machine learning for Geospatial Interlinking. They stick to a static processing order, except when the algorithm presented in [18] is applied. When two geometries s and t are related, their weight w is updated using the formula $w' = w \times (1 + q)$, where q tracks their relatedness frequency. However, Dynamic Scheduling has a limited scope and, thus, we disregard it in this work.

3.2 Supervised GIA.nt

SupervisedGIAnt" operates by employing a technique called "Supervised Scheduling" [25] instead of the traditional "Scheduling" method, as depicted in Figure 2. Supervised Scheduling performs probabilistic binary classification, assigning to each candidate pair a probability that is proportional to the likelihood that its constituent geometries are topologically related.

The algorithm starts by calculating the dimensions of grid cells based on specific properties of the source dataset's width and height. It then proceeds to index by identifying the tiles overlapping with the Minimum Bounding Rectangle (MBR) of each source geometry. For training set generation, pair IDs are randomly

selected within a specified range. For each target geometry, tiles intersecting with its MBR are determined, and source geometries are aggregated into a candidate set. Candidate-based features are updated, and pairs are added to a random sample for verification. The sampled pairs are shuffled, and then verified and classified as topologically related or not. 31 Feature vectors are generated for the sampled pairs. The training set is then fed into a chosen algorithm to learn the classification model. Iterating over the target dataset involves gathering source candidates from intersecting MBR tiles, generating feature vectors, and predicting classification probabilities for each pair. Pairs exceeding a certain probability threshold are added to a priority queue, with the threshold updated if necessary. Verification of top-weighted pairs occurs next, with their topological relations added to the set of links. Finally, space and time complexity are analyzed, encapsulating various steps such as indexing, training set generation, model training, candidate selection, feature vector generation, and verification, ultimately leading to the identification of topologically related geometries.

4 RECALL-CONSTRAINED TOPOLOGICAL RELATION DETECTION MODEL

Our goal in this research endeavor is to strike a delicate balance between precision and computational efficiency in identifying topological relations within spatial data. It introduces the concept of Recall-Constrained Topological Relation Detection (RCTRD), which prioritizes identifying significant relations while optimizing computational resources. RCTRD aims to strike a balance between accuracy and efficiency by setting a predetermined level of recall, acknowledging the practical constraints of large-scale datasets. It outlines the challenges and objectives of RCTRD, emphasizing the need for innovative algorithms and methodologies to address these challenges effectively. Notably, RCTRD addresses the inherent difficulty of not knowing the total number of related pairs in input datasets, requiring solutions that either predict this number or leverage heuristics to overcome this limitation.

4.1 Extrapolation Algorithm

The core idea of this algorithm is to infer the number of duplicates in the given dataset from a random, but representative sample of the candidate pairs. To this end, it verifies a limited a limited number of random candidate pairs in order to assess the portion of qualifying ones. This portion is then extrapolated to the entire set of valid candidate pairs, i.e., source and target geometries with intersecting MBRs. To retain the minimum memory requirements of GIA.nt, this can only be accomplished by reading the target geometries from the disk twice: once for determining the total number of candidate pairs and selecting a random sample and once for verifying the random sample and selecting the top-weighted pairs. In this way, this approach doubles the run time of Scheduling in order to minimize the run-time of Verification.

Initially, the algorithm utilizes GIA.nt's method to filter candidate pairs based on intersecting Minimum Bounding Rectangles (MBRs). For the sampling phase, the algorithm opens a reader to sequentially load target geometries from disk into memory. It retrieves source geometries that intersect the MBR(t) for each target geometry, incrementing the total number of candidate pairs when such intersections occur. After processing all target geometries, the algorithm selects N random pair IDs from [1, $totalCP$], where $totalCP$ is the total number of candidate pairs. Following sampling, the algorithm moves on to scheduling, reopening the target dataset reader to commence processing. For each target geometry, candidate source geometries are retrieved from the index and checked for intersection with the corresponding MBR. Upon identifying a candidate pair, the algorithm assigns the next ID and proceeds to verify it. If the ID is among the randomly selected ones, the algorithm verifies the pair, increments the counter $detectedQP$, adds topological relations to the output, and includes the pair in a main memory hash set V to prevent redundant examination. Additionally, the algorithm weights all candidate pairs and adds them to a priority queue PQ. Finally, Scheduling computes the maximum number of verifications that will be carried out during the next step as follows.

$$maxVerif\ ications = re_d \cdot \frac{detectedQP}{N} \cdot totalCP$$

where $re_d \in (0, 1)$ denotes the desired recall level. If $maxVerifications$ is not zero, the algorithm proceeds to verification. In the final verification stage, the algorithm iterates over the top-weighted pairs in the priority queue. For each pair, the algorithm examines it and adds any topological links to the output if not yet verified. If the pair qualifies, the algorithm increments the counter. Processing continues until the counter reaches $maxVerifications$ or the priority queue is exhausted, signaling the termination of processing.

Note that the Algorithm describes the simplest case of the extrapolation algorithm, which uses Unsupervised Scheduling, i.e., an unsupervised weighting scheme. In case Supervised Scheduling is employed, the sampling stage also performs the necessary feature estimations, while using the verified sample of geometries as a labelled dataset for training the binary probabilistic classifier. In our experiments we used Supervised Scheduling.

4.2 Heuristics Algorithm

We now present the simplest family of algorithms that try to exceed the desired recall level in an implicit way. To this end, they iterate over all candidate pairs just once, placing the top-weighted ones in a priority queue. The priority queue is verified in descending weight. After every Verification, a heuristic condition is checked to decide whether to terminate the entire processing or not.

Initially, the same Filtering as Progressive GIA.nt is applied. This is followed by the same Scheduling process as Progressive GIA.nt: a reader is opened to read the target dataset on the fly. For each target geometry t, we retrieve the set of source geometries, CS, which are contained in the tiles intersecting t's MBR. CS is filtered to retain only those source geometries that indeed intersect t's MBR. The retained pairs are weighted according to the given weighting scheme and are inserted into the priority queue with the top-weighted pairs, PQ. After closing the target dataset reader, the Verification step starts. For the next top-weighted candidate pair in PQ, the algorithm computes the intersection matrix of its constituent geometries and extracts the ensuing set of topological relations, l. If l is not empty, the pair is a qualifying one; after merging l with the output L, we

examine the predetermined heuristic condition. If the condition is satisfied, the algorithm terminates.

We observe that the performance of the algorithm depends exclusively on the heuristic termination condition. To thoroughly test its potential, we came up with the following versatile conditions:

1) *Precision Threshold*. Verification terminates as soon as precision falls below a specific level. The higher the desired recall is, the lower this precision threshold should be and vice versa.
2) *Qualifying Distance Threshold*. Verification terminates as soon as the number of unrelated, but verified pairs that intervene between two consecutive qualifying ones exceeds a certain limit. The higher the desired recall level is, the larger this threshold should be.
3) *Dynamic Qualifying Distance Threshold*. This heuristic converts the above static threshold into a dynamic one. Instead of a predetermined distance, the number of allowed unrelated Verifications increases as more qualifying pairs are detected. The higher the desired recall level is, the larger the increment in the distance threshold should be.
4) *Buffered Threshold*. This heuristic extends all the above ones with a buffer that allows a specific number of violations of the termination condition.

The drawback of these heuristics is that they are indirectly related to the desired recall level. This is because they are independent of any estimation of the actual number of qualifying pairs in the given datasets. Yet, this shortcoming can be counterbalanced by background knowledge in the form of qualitative estimation of the portion of qualifying pairs over the set of candidate pairs, when these heuristics are configured by human experts. Note that the Algorithm we described assumes an Unsupervised Scheduling step, which corresponds to the simplest case. In practice, though, Supervised Scheduling can be used, too. This is done by inserting a sampling phase between Filtering and Scheduling, which performs the necessary features and builds a random labeled dataset for training the binary probabilistic classifier.

4.3 KDE Based Algorithm

This chapter explores a more complex, principled and innovative approach to streamline the verification process within the Supervised Progressive Scheduling framework. The methodology integrates Kernel Density Estimation (KDE) into the existing system. By focusing on probabilistic predictions and estimation techniques, this approach aims to significantly reduce the number of verifications necessary while ensuring the desired recall level.

The functionality of this approach begins by implementing filtering and training stages, akin to the Supervised Progressive GIA.nt process. The filtering step refines the dataset, while the training phase constructs a probabilistic binary classification model, laying the foundation for the subsequent steps.

A subset of candidate geometries is randomly chosen for classification by the pre-trained classifier. Verification determines the topological relationships for each pair, establishing a labeled dataset. This labeled set of instances with classification probability and class labels serves as the foundation for training the Kernel Density Estimator (KDE). Various KDE models are explored, emphasizing different techniques but we omit these details for brevity. The optimal KDE model is selected through an evaluation, which estimates the most effective KDE approach in the given labeled dataset. Using the trained KDE model, we estimate the recall level that corresponds to different classification probabilities. This estimation guides the selection of the minimum probability threshold required to achieve the desired recall level.

Subsequently, the probabilistic classifier calculates the probability for each pair of candidate geometries i.e., for each pair of geometries with intersecting MBRs. Pairs exceeding the minimum probability threshold are subject to verification, while the rest are discarded, thus reducing the verification workload significantly.

More specifically, for each target geometry t, we retrieve the set of source geometries, CS which are contained in the tiles intersecting t's MBR . CS is filtered to retain only those source geometries that intersect t's MBR. The trained probabilistic classifier calculates the probability for each retained pair of candidate geometries as in SupervisedGIAnt. Pairs exceeding the minimum

probability threshold, which has been determined by the trained KDE model, are subject to verification. This step significantly reduces the verification workload.

After closing the target dataset reader, the Verification step starts. For the next candidate pair in Ts, the algorithm computes the intersection matrix of its constituent geometries and extracts the ensuing set of topological relations, IM. If IM is not empty, the pair is a qualifying one and IM is merged with the output LR. After merging IM with the output LR the algorithm terminates as soon as all retained pairs in TC are verified.

The integration of Kernel Density Estimation into the Supervised Progressive Scheduling framework offers a promising avenue for enhancing efficiency while ensuring accuracy. By relying on probabilistic predictions and estimation, the algorithm minimizes verifications, making it a valuable tool in geographic information systems and spatial data processing

5 EVALUATION

Our experiments rely on subsets of publicly available, large-scale, real-world datasets that are popular in the literature [7, 17, 24] and involve LineStrings and Polygons. Their technical characteristics are presented in Table 1. We observe in all cases that the number of qualifying pairs, which is equal to #Intersects, accounts for a tiny portion of the Cartesian product between the source and the target geometries. This suggests that the overall computational cost can be reduced by orders of magnitude in comparison to the brute force approach.

Table 1: Technical characteristics of the datasets used in our experiments. Note that in each dataset, the number of qualifying pairs is equal to #intersects.

	s1	s2	s3	s5	s7
#Source geometries	229,276	458,552	458,553	1,146,383	1,604,936
#Target geometries	583,833	1,167,667	1,751,501	2,919,169	4,086,837
#contains	37,217	151,620	124,461	378,688	576,101
#covered-by	0	0	0	0	0
#covers	39,173	157,416	129,246	388,453	594,117
#crosses	38,994	159,314	131,464	388,380	605,198
#equals	0	0	0	0	0
#intersects	127,939	424,155	403,518	967,765	1,836,670
#overlaps	1,777	7,694	7,003	9,692	29,097
#touches	88,945	264,841	272,054	579,385	1,231,472
#within	0	0	0	0	0
Total Topological Relations	334,045	1,165,040	1,067,746	2,712,363	4,872,655

5.1 Experiment 1: Evaluating the KDE Approach Algorithm and Baseline Methods

This subsection provides a thorough evaluation of the KDE Approach Algorithm using various heuristic functions to achieve desired recall levels of 0.3, 0.5, and 0.7. Heuristic functions explored include Precision Threshold, Qualifying Distance, and Dynamic Qualifying Distance. Each heuristic was rigorously tested to understand its impact on algorithm performance. We first check our results when the user gives the desired recall level 0.5.

Precision Threshold Heuristic: The Precision Threshold heuristic demonstrated balanced behavior. Setting the threshold at 1.0, the algorithm exhibited best performance, establishing an equilibrium between recall and precision. Testing it with dataset $s2$ a precision of 0.5 indicated the best validity of the identified pairs, minimizing false positives while ensuring a reasonable recall (of 0.824).

Qualifying Distance Heuristic: Qualifying distance, as spatial parameter, played a crucial role in the algorithm's decision-making process. Experiments revealed that a qualifying distance of 100 yielded the most satisfactory results. This parameter emphasized the algorithm's sensitivity to spatial proximity. By focusing on pairs within a close spatial range, the algorithm efficiently reduced irrelevant verifications, optimizing its efficiency.

Dynamic Qualifying Distance Heuristic: The Dynamic Qualifying Distance heuristic introduced adaptability to the evaluation. By coupling a qualifying distance of 10

with a dynamic factor of 1.0, the algorithm adjusted its qualifying distance based on evolving dataset characteristics. The recall remained consistently high at 0.819, while precision held at 0.57, reflecting the algorithm's ability to adapt to varying dataset densities.

Additional Experiments: Further evaluations at recall levels of 0.3 and 0.7 revealed the algorithm's scalability and adaptability. At lower recall levels, precision-oriented strategies were observed, while broader search approaches were employed at higher recall levels, showcasing the algorithm's ability to balance precision and recall.

5.2 Experiment 2: Evaluating the Extrapolation Algorithm

This experiment focuses on the performance of the "Extrapolation Algorithm" with supervised scheduling under varying conditions, represented by sample sizes of $N = 100$, $N = 1000$, and $N = 10000$.

The algorithm's precision, recall, and computational efficiency were analyzed for different sample sizes, highlighting trade-offs and optimal performance. Factors such as precision versus recall trade-offs, computational efficiency, and the optimal choice of sample size were discussed, with $\mathrm{N}{=}10000$ demonstrating the best balance between accuracy and efficiency.

5.3 Experiment 3: Comparison of SupervisedGIA.nt and KDE-Based Algorithm

A comparison between SupervisedGIA.nt and the KDE-Based Algorithm revealed superior recall and verification efficiency of the latter, making it a promising choice for applications where accuracy and computational efficiency are critical.

5.4 Experimental Analysis: KDE Based Algorithm vs Extrapolation Algorithm

A comparative study favored the KDE-Based Algorithm for its superior verification time and memory efficiency, making it a compelling choice for real-world spatial data processing tasks.

This chapter provides a comprehensive evaluation of spatial data processing algorithms, highlighting their strengths, weaknesses, and practical implications for real-world applications.

6 CONCLUSIONS AND FUTURE WORKS

In the realm of spatial data analysis, striking a balance between precision and computational efficiency is of paramount importance. This thesis explores innovative algorithms in Recall-Constrained Topological Relation Detection, each with its unique strengths and limitations. Advantages of the Extrapolation Algorithm include the efficient use of computational resources by verifying a limited number of random candidate pairs while It also provides a representative subset of topological relations within the dataset. The Heuristics Algorithm offers high time efficiency by iterating over candidate pairs only once and has versatile termination conditions, allowing adaptability to different requirements. The KDE Based Algorithm employs a probabilistic approach, balancing precision and efficiency by utilizing Kernel Density Estimation while Its customizable thresholds and integration with existing frameworks make it a standout choice for real-world applications.

Looking ahead, this chapter highlights potential avenues for future research in spatial data analysis. Enhanced probabilistic models, dynamic heuristics tailored to dataset characteristics, and exploration of real-time applications represent just a few promising directions. Additionally, integrating with spatial databases, analyzing multi-modal data, and considering ethical implications and biases in algorithmic decisions offer fertile ground for exploration.

By embracing these future directions, we can continue to push the boundaries of spatial data analysis, offering more accurate, efficient, and ethically sound solutions, paving the way for transformative developments in spatial data analysis techniques.

REFERENCES

[1] Abdullah Fathi Ahmed, Mohamed Ahmed Sherif, and Axel-Cyrille Ngonga Ngomo. 2018. RADON2 - a buffered-intersection matrix computing approach to accelerate link discovery over geo-spatial RDF knowledge bases. In OAEI.

[2] Edward P. F. Chan and Jimmy N. H. Ng. 1997. A General and Efficient Implementation of Geometric Operators and Predicates. In SSD, Vol. 1262. 69–93.

[3] Eliseo Clementini, Paolino Di Felice, and Peter van Oosterom. 1993. A Small Set of Formal Topological Relationships Suitable for End-User Interaction. In SSD.

[4] Eliseo Clementini, Jayant Sharma, and Max J. Egenhofer. 1994. Modelling topological spatial relations: Strategies for query processing. Comput. Graph. (1994).

[5] Alishiba Dsouza et al. 2021. WorldKG: A World-Scale Geographic Knowledge Graph. In CIKM. 4475–4484.

[6] Max J Egenhofer and Robert D Franzosa. 1991. Point-set topological spatial relations. International Journal of Geographical Information System 5, 2 (1991).

[7] Ahmed Eldawy and Mohamed F. Mokbel. 2015. SpatialHadoop: A MapReduce framework for spatial data. In ICDE. 1352–1363.

[8] Eibe Frank, Mark Hall, and Ian Witten. 2016. The WEKA Workbench. Online Appendix for" Data Mining: Practical Machine Learning Tools and Techniques.
https://www.cs.waikato.ac.nz/ml/weka/Witten_et_al_2016_appendix.pdf

[9] Jiawei Han, Micheline Kamber, and Jian Pei. 2011. Data Mining: Concepts and Techniques, 3rd edition. Morgan Kaufmann.

[10] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. Artif. Intell. 194 (2013), 28 61.

[11] Krzysztof Janowicz et al. 2022. Know, Know Where, Knowwheregraph: A Densely Connected, Cross-Domain Knowledge Graph and Geo-Enrichment Service Stack for Applications in Environmental Intelligence. AI Mag. 43, 1 (2022), 30–39.

[12] Nikolaos Karalis, Georgios Mandilaras, and Manolis Koubarakis. 2019. Extending the YAGO2 Knowledge Graph with Precise Geospatial Knowledge. In ISWC.

[13] Oje Kwon and Ki-Joune Li. 2011. Progressive spatial join for polygon data stream. In SIGSPATIAL. 389–392.

[14] Rushi Longadge and Snehalata Dongre. 2013. Class Imbalance Problem in Data Mining Review. CoRR abs/1305.1707 (2013).

[15] Nikos Mamoulis. 2011. Spatial data management. Synthesis Lectures on Data Management 3, 6 (2011), 1–149.

[16] Axel-Cyrille Ngonga Ngomo. 2013. ORCHID - Reduction-Ratio-Optimal Computation of Geo-spatial Distances for Link Discovery. In ISWC. 395–410.

[17] George Papadakis, Georgios Mandilaras, Nikos Mamoulis, and Manolis Koubarakis. 2021. Progressive, Holistic Geospatial Interlinking. In WWW.

[18] George Papadakis, George Mandilaras, Nikos Mamoulis, and Manolis Koubarakis. 2022. Static and dynamic progressive geospatial interlinking. ACM Transactions on Spatial Algorithms and Systems (TSAS) 8, 2 (2022), 1–41.

[19] Georgios M. Santipantakis, Apostolos Glenis, Christos Doulkeridis, Akrivi Vlachou, and George A. Vouros. 2019. stLD: towards a spatio-temporal link discovery framework. In SBD@SIGMOD. 4:1–4:6.

[20] Peter Sbarski and Sam Kroonenburg. 2017. Serverless architectures on AWS: with examples using Aws Lambda. Simon and Schuster.

[21] Mohamed Ahmed Sherif, Kevin Dreßler, Panayiotis Smeros, and Axel-Cyrille Ngonga Ngomo. 2017. Radon - Rapid Discovery of Topological Relations. In AAAI. 175–181.

[22] Panayiotis Smeros and Manolis Koubarakis. 2016. Discovering Spatial and Temporal Links among RDF Data. In Workshop on Linked Data on the Web, LDOW.

[23] Wee Hyong Tok, Stéphane Bressan, and Mong-Li Lee. 2006. Progressive Spatial Join. In SSDBM. 353–358.

[24] Dimitrios Tsitsigkos, Panagiotis Bouros, Nikos Mamoulis, and Manolis Terrovitis. 2019. Parallel In-Memory Evaluation of Spatial Joins. In SIGSPATIAL. 516–519.

[25] Maria Despoina Siampou, George Papadakis, Nikos Mamoulis, and Manolis Koubarakis. 2023. Supervised Scheduling for Geospatial Interlinking. In The 31st ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL '23), November 13-16, 2023, Hamburg, Germany. ACM, New York, NY, USA, 12 pages.

[26] Manolis Koubarakis (Editor), 2023. GEOSPATIAL DATA SCIENCE: A HANDS-ON APPROACH FOR BUILDING GEOSPATIAL APPLICATIONS USING LINKED DATA TECHNOLOGIES, 1st Edition. Publisher: Association for Computing Machinery.

[27] George Papadakis. 2023. Interlinking Geospatial Data Sources. In GEOSPATIAL DATA SCIENCE: A HANDS-ON APPROACH FOR BUILDING GEOSPATIAL APPLICATIONS USING LINKED DATA TECHNOLOGIES, 1st Edition. Publisher: Association for Computing Machinery. 161-184, https://doi.org/10.1145/

[28] Marios Papamichalopoulos, George Papadakis, George Mandilaras, Maria Despoina Siampou, Nikos Mamoulis, Manolis Koubarakis. 2022. Three-dimensional Geospatial Interlinking with JedAI-spatial. In CoRR, Vol. abs/2205.01905, https://doi.org/10.48550/

Machine Learning Snowfall Retrieval Algorithms for Satellite Precipitation Estimates

Ioannis Th. Dravilas

ABSTRACT

Remote sensing of snowfall has been proved to be a significant challenge since the start of the satellite era. Several techniques have been applied to satellite data, in order to estimate the fraction of frozen precipitation that reaches the surface. This thesis aims at investigating the efficacy of different Machine Learning (ML), and especially Deep Learning (DL) algorithms, in estimating the precipitation phase of NASA's Integrated Multi-satellitE Retrievals for the Global Precipitation Measurement (GPM-IMERG). To achieve that, a training phase with hourly high-resolution numerical model outputs and in-situ observational data is chosen for the period of late-2020 and 2021. Results show that ML and DL models can estimate precipitation phase with relatively high accuracy, when compared to traditional methods, based on several case studies. The findings suggest that ML models offer a promising approach for advancing the nowcasting of snowfall and building a long-term archive dataset of IMERG-based snowfall, utilizing conventional near real-time data.

ADVISOR

Manolis Koubarakis, Professor NKUA

1 INTRODUCTION

1.1 Precipitation

Precipitation is defined as "all liquid or solid phase aqueous particles that originate in the atmosphere and fall to the Earth's surface" [1].

1.2 Precipitation Phase

Deriving the phase of precipitation and distinguishing between its liquid and frozen state, is of major importance for human activities, hydrological processes and climate change studies [2].

Towards this direction, a plethora of techniques is being used today to detect snowfall. Some of the most successful methods include using in-situ observations, remote sensing through dual-wavelength radars, or using numerical weather models [3]. However, none of those approaches has been proved fully reliable, while for the most accurate ones, such as the measurements from in-situ instruments, the available data are generally sparse or even absent, for example in mountainous or sparsely populated areas [2, 4]. The use of satellite data to obtain precipitation estimates has been one of the most used methods for measuring precipitation so far, giving both satisfactory and continuous results with almost no missing values or temporal and spatial gaps [5].

1.3 Machine Learning

Rebala et al. (2019) [6] described Machine Learning (ML) as a computer science field focusing on enabling computers to learn and improve their performance without requiring explicit programming. Traditional programming methods involve creating a detailed design and implementing it as a program, but this can be challenging for problems like detecting handwritten characters, due to the difficulty in designing rules for such variations.

ML can be applied to various types of problems such as: classification, where data are categorized into different classes, like will it rain or snow tomorrow;

clustering, where data points are grouped into clusters based on shared properties; and regression, which uses historical data to forecast a continuous range of values, like temperature.

The use of Machine Learning in Meteorology has been constantly increasing during the last few years (Chase et al. 2022) [7], and is predicted to increase even more, as the volume of meteorological data that can be used to train ML models grows.

2 BACKGROUND AND RELATED WORK

The problem of deriving precipitation phase has been extensively studied before, both with conventional and Machine Learning methods.

2.1 Conventional Methods

Matsuo et al. in 1981 [8] showed that liquid water content and fall velocity of snowflakes, and therefore the depth of the layer below freezing level where melting does not occur, were dependent on surface air temperature, relative humidity, and snowflake mass.

Sims and Liu in 2015 [2] developed a parameterization scheme that utilizes 2-m temperature, relative humidity, low-level vertical lapse rate, surface skin temperature and surface type to calculate the conditional probability of solid precipitation to occur. Surface pressure is also used in order to calculate wet-bulb temperature (Tw).

2.2 Machine Learning Methods

In 2018, Behrangi et al. [4] reported that near-surface air temperature is usually used to derive precipitation phase. It was found that relative humidity, wind speed and air pressure can also affect the melting of snowflakes and thus more variables should be used together with air temperature to determine precipitation phase. Even though among all single predictors Tw yields the

highest score, the authors concluded that the use of logistic regression to combine the previously mentioned variables produces even better results.

A paper presented at the AGU Fall Meeting 2022 by Bédard-Therrien et al. [9] introduced an ML approach for partitioning precipitation phase, using air temperature, relative humidity and disdrometer data, along with a Random Forest Regression Model.

3 MATERIALS AND METHODS

In this study, Machine Learning and especially Deep Learning (DL) algorithms are used along with numerical weather data and in-situ observational data to classify the phase of precipitation acquired by the Integrated Multi-satellitE Retrievals for the Global Precipitation Measurement (GPM-IMERG) operated by the National Aeronautics and Space Administration (NASA) [10].

3.1 Description of the Acquired Data

3.1.1 In-situ Observations From Ground Stations

During the past 15 years, the Institute for Environmental Research and Sustainable Development of the National Observatory of Athens (NOA/METEO) has established and is currently managing a dense network of automated weather stations throughout Greece (NOAAN) [11].

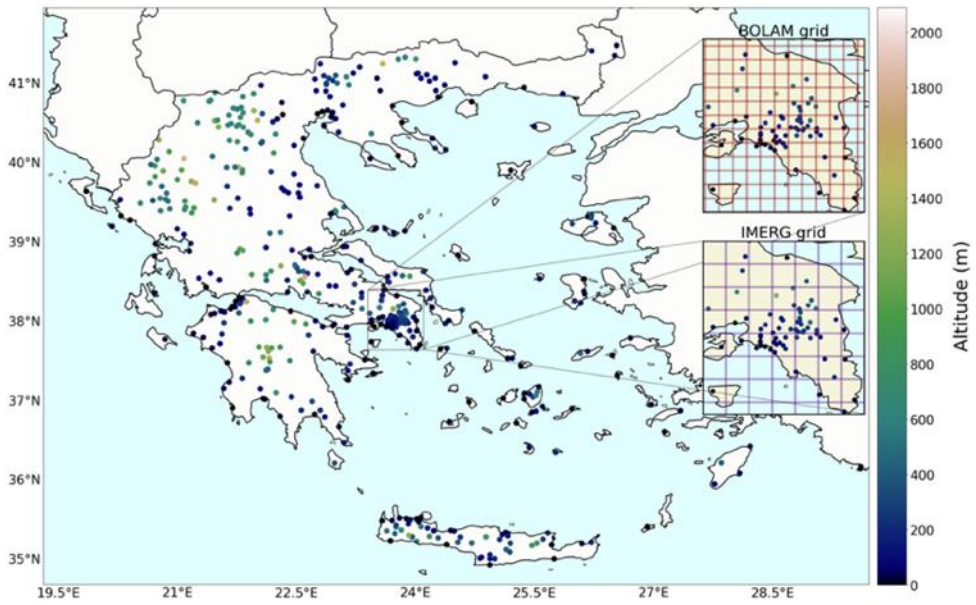


Figure 1: The NOAAN weather stations used, coloured by altitude, and the corresponding grids of BOLAM and IMERG over the Attica region

3.1.2 Numerical Weather Model Data

The National Observatory of Athens also runs the hydrostatic meteorological Bologna Limited-Area Model (BOLAM) in operational mode [12].

3.1.3 Satellite Precipitation Estimates

The Integrated Multi-satellitE Retrievals for GPM (IMERG) algorithm is designed to intercalibrate, combine and interpolate microwave precipitation measurements, along with microwave calibrated infrared (IR) satellite measurements, precipitation gauge analyses and possibly other precipitation estimators, at fine time and space scales worldwide [10].

Precipitation phase in IMERG is currently computed diagnostically, based on the Liu scheme [2]. The Liu scheme used by NASA calculates the Probability of Liquid Precipitation Phase (PLPP) based solely on data from a numerical weather model or model analysis, relying on surface wet-bulb temperature values.

3.2 Creation of a Custom Dataset

Using NOAAN observations for air temperature, relative humidity and atmospheric pressure, BOLAM's nowcast (namely the first 12 hours after model initialization time excluding a spin-up period of 12 hours) and the 1.1°C Tw threshold chosen by NASA for IMERG V06 over land as the value corresponding to a Probability of Liquid Precipitation Phase equal to 50%, a new dataset is created. For each in-situ observation, the new dataset contains information about whether conditions were favorable for snowfall according to the surface Tw<1.1°C threshold, the corresponding numerical weather model data for the nearest grid point, as well as the station metadata such as latitude, longitude and altitude. The dataset comprises data collected from 480 locations across Greece, covering the time period of late-2020 and 2021. The temporal resolution of the dataset is set at 30 minutes, providing detailed and frequent measurements that match the temporal resolution of IMERG V06.

3.3 Machine Learning Models Used

3.3.1 Random Forest

Random Forest was first introduced in 2001 and is consisted of many basic classifiers in the form of Decision Trees [13]. Each Decision Tree makes a prediction, which is completely independent from the decisions of the other Decision Trees, and, in classification tasks, the final result is produced by a voting procedure, resembling an ensemble technique.

3.3.2 Gradient Boosting

Gradient Boosting is a popular ML technique used, among others, in classification tasks. It works by creating multiple weak models, which often are Decision Trees, and combining them to form a better-performing model. This is usually done by building an initial weak model, then a second model aiming to more accurately predict the cases where the first one performs poorly, etc. Each new model created, targets minimizing the error of the loss function; thus, the gradient of the loss function is calculated in every step of the algorithm [14].

3.3.3 Feedforward Neural Network

Feedforward Neural Networks are the simplest type of artificial neural networks, where information moves only in a forward direction, from the input nodes, to the hidden nodes and to the output nodes [15]. Here, a multi-layer neural network, also called a Multilayer Perceptron is used. A Multilayer Perceptron consists of multiple layers of computational units, containing neurons that are connected to the neurons of the next layer. These models are trained using back-propagation, a technique utilized to adjust the weight values of each connection, in a way that minimizes the error between predictions and actual values [16].

3.4 Training And Testing Process

3.4.1 Data Splitting and Cross-Validation

Data are divided into training and testing datasets based solely on station locations, using an 80:20 ratio.

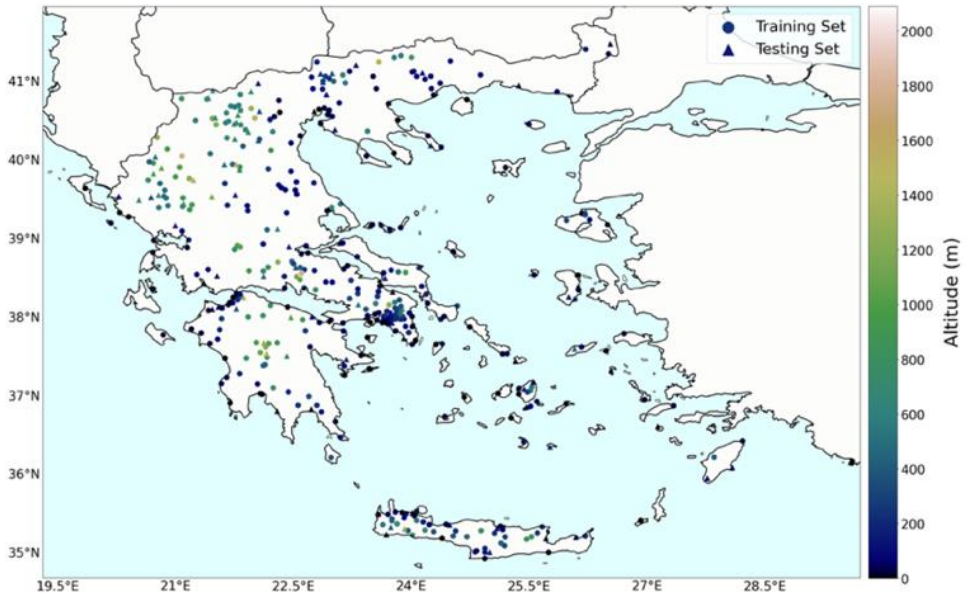


Figure 2: The NOAAN weather stations divided into the training and the testing dataset, coloured by altitude

For each of the three models described above, the best architecture is determined through a 5-fold cross-validation process for hyperparameter

tuning, ensuring that data for each station location are exclusively present in only one of the training and validation datasets during each iteration.

Finally, the trained models are tested on the corresponding testing dataset, comprising 20% of the initial dataset.

3.5 Evaluation and Metrics

The metrics used to evaluate the results of the models are Precision, Recall (also called Probability of Detection - POD), F1-score, Critical Success Index (CSI), False Alarm Ratio (FAR) and Heidke Skill Score (HSS).

4 RESULTS

4.1 Best Hyperparameters

For each of the Random Forest, Gradient Boosting and Feedforward Neural Network models, the best architecture is selected after hyperparameter tuning, and is then evaluated on the testing dataset.

4.2 Feature Importance

Notably, both Temperature at 2 m and altitude emerge as highly influential variables, making appearances in both types of importance metrics. Additionally, Temperature and Specific Humidity at the 1000 hPa isobaric surface stand among the top three features, completing the groups of the three most important contributors.

4.3 Evaluation on the First Testing Dataset

The testing dataset used here is the one containing data for late-2020 and 2021 from 96 station locations in a 30-minute time-step.

4.3.1 Machine Learning Models

The six score metrics were calculated on the testing dataset for all the Random Forest, Gradient Boosting and Feedforward Neural Network models, with regard to their ability of predicting cases with conditions favorable for snowfall (Tw<1.1°C).

Table 4.1: Scores of the 3 ML models for predicting cases with conditions favorable for snowfall, evaluated on the first testing dataset.

Model	Precision	Recall (POD)	F1-score	CSI	FAR	HSS
Random Forest	0.87	0.72	0.79	0.65	0.13	0.78
Gradient Boosting	0.87	0.81	0.84	0.72	0.13	0.83
Feedforward Neural Network	0.85	0.80	0.82	0.70	0.15	0.81

4.3.2 Conventional Methods

A comparison with traditional precipitation phase derivation techniques is also made on the same testing dataset.

Table 4.2: Scores of the 2 conventional methods for predicting cases with conditions favorable for snowfall, evaluated on the first testing dataset.

Model	Precision	Recall (POD)	F1-score	CSI	FAR	HSS
IMERG V06 PLPP	0.61	0.61	0.61	0.44	0.39	0.59
BOLAM nowcast	0.66	0.81	0.73	0.57	0.34	0.71

4.4 Application and Evaluation on 2022 Data

In this section, a new dataset comprising data from the initial three months of 2022 is introduced.

4.4.1 Examples of Application on IMERG Precipitation Estimates

During the initial three months of 2022, Greece experienced several cold waves of varying intensity. To assess the effectiveness of the Gradient Boosting model in identifying the precipitation phase during these events, BOLAM nowcast data are used as input. The precipitation phase predicted by the model is then utilized to mask the corresponding IMERG V06 Early Run uncalibrated precipitation data for the same time period.

4.4.1.1 10th of January 2022

On January 10th of 2022, a cut-off low in the upper/mid troposphere moved from Italy towards the Ionian Sea in Greece. This weather system was accompanied by a mild cold air advection from the Balkans towards Northern Greece. As a result of these weather conditions, snowfall was expected in the mountains of Mainland Greece and in some lower altitude areas of Western Macedonia.

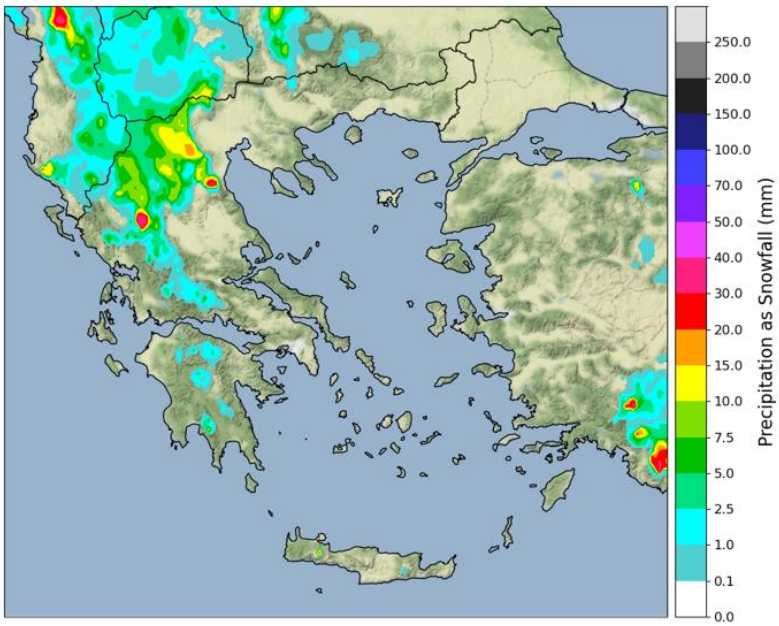


Figure 3: The 24-hour accumulated precipitation from IMERG that fell as snowfall during the 10th of January 2022 in Greece, as indicated by the Gradient Boosting model.

4.4.1.2 24th of January 2022

On January 24th of 2022, an upper level closed low over the Eastern Mediterranean resulted in a significant cold air advection, leading to heavy snowfall in the eastern regions of Greece, including Attica and the capital city, Athens, as well as the Aegean Islands.

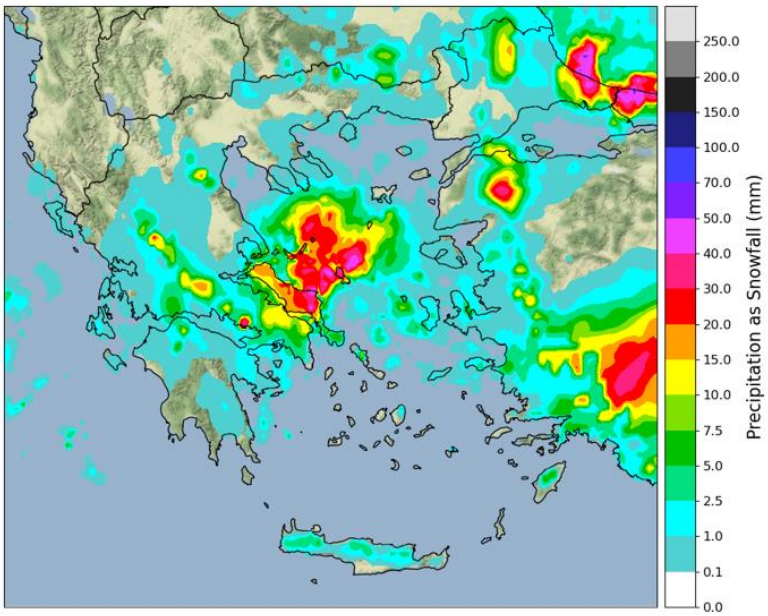


Figure 4: The 24-hour accumulated precipitation from IMERG that fell as snowfall during the 24th of January 2022 in Greece, as indicated by the Gradient Boosting model.

5 DISCUSSION

5.1 Comparison With Previous Work

Moon et al. in 2020 [17] achieved an HSS of 73% in determining precipitation type for snow cases, using ML models trained with short-range forecasts from numerical models). The individual HSS values for ECMWF and RDAPS alone were comparatively lower at 52% and 55% respectively, while the improved Matsuo scheme [18] used operationally at the time by the Korea Meteorological Administration (KMA) exhibited an HSS of 71%.

In 2022, Xiong et al. [19] evaluated IMERG and ERA5 precipitation phase partitioning on a global scale, using target data from ground observations. POD of snowfall over land was 87% for IMERG and 91% for ERA5. CSI was 67% for IMERG and 81% for ERA5, while FAR was 16% for IMERG and 7% for ERA5.

The Gradient Boosting model used in this study achieved a POD of 81%, CSI of 72%, HSS of 85% and FAR of 13%. This is a significant improvement compared to most of the previously-described methods.

6 CONCLUSIONS

During this study an algorithm that is able to identify the precipitation phase of IMERG precipitation data was developed, leveraging Machine Learning models based on Random Forest and Gradient Boosting, and a Deep Learning model employing a Feedforward Neural Network. The 1.1°C wet-bulb temperature was used as an upper threshold for solid precipitation to occur over land. The results of our analysis indicate that the use of Machine Learning models is a very promising approach for estimating precipitation phase. Specifically, it was found that 81% of the actual snow-favorable conditions can be identified, while 87% of all the predicted snow-favorable conditions are proved correct. Application of the best-performing model's output on IMERG precipitation estimates from real-world cases, also shows that rain-snow partitioning on IMERG data yields comprehensive and reliable results.

The developed model's capability to accurately determine precipitation phase on satellite data, holds tremendous potential for near-real-time snowfall monitoring, providing valuable insights for emergency responses and aid distribution in areas affected by severe weather. Furthermore, this model opens up new possibilities for creating a thorough and enduring snowfall dataset, significantly enhancing our understanding of hydrological processes, supporting various water resource management initiatives and contributing to a deeper understanding of climate change impacts.

While our study has several strengths, it is not without limitations. For example, the wet-bulb temperature threshold applied to distinguish between solid and liquid precipitation on in-situ observational data, is not the optimal indicator for the actual precipitation phase. It is planned to make use of additional snowfall in-situ data from NOAAN in order to further evaluate the developed models.

REFERENCES

[1] S. Michaelides, V. Levizzani, E. Anagnostou, P. Bauer, T. Kasparis, and J. Lane, "Precipitation: Measurement, remote sensing, climatology and modeling," Atmospheric Research, vol. 94, no. 4, pp. 512–533, Dec. 2009.

[2] E. M. Sims and G. Liu, "A Parameterization of the Probability of Snow–Rain Transition," Journal of Hydrometeorology, vol. 16, no. 4, pp. 1466–1477, Jul. 2015.

[3] L. Liao, R. Meneghini, T. Iguchi, and A. Detwiler, "Use of Dual-Wavelength Radar for Snow Parameter Estimates," Journal of Atmospheric and Oceanic Technology, vol. 22, no. 10, pp. 1494–1506, Oct. 2005

[4] A. Behrangi, X. Yin, S. Rajagopal, D. Stampoulis, and H. Ye, "On distinguishing snowfall from rainfall using near-surface atmospheric information: Comparative analysis, uncertainties and hydrologic importance," Quarterly Journal of the Royal Meteorological Society, vol. 144, no. S1, pp. 89–102, Aug. 2018

[5] R. K. Pradhan, Y. Markonis, M. R. V. Godoy, A. Villalba-Pradas, K. M. Andreadis, E. I. Nikolopoulos, S. M. Papalexiou, A. Rahim, F. J. Tapiador, and M. Hanel, "Review of GPM IMERG performance: A global perspective," Remote Sensing of Environment, vol. 268, p. 112754, Jan. 2022.

[6] G. Rebala, A. Ravi, and S. Churiwala, "Machine Learning Definition and Basics," in An Introduction to Machine Learning. Springer International Publishing, 2019, pp. 1–17.

[7] R. J. Chase, D. R. Harrison, A. Burke, G. M. Lackmann, and A. McGovern, "A Machine Learning Tutorial for Operational Meteorology. Part I: Traditional

Machine Learning," Weather and Forecasting, vol. 37, no. 8, pp. 1509–1529, Aug. 2022

[8] T. Matsuo and Y. Sasyo, "Melting of Snowflakes below Freezing Level in the Atmosphere," Journal of the Meteorological Society of Japan. Ser. II, vol. 59, no. 1, pp. 10–25, 1981.

[9] D. Nadeau, "Operational Partitioning of Precipitation Phase Using Machine Learning — ui.adsabs.harvard.edu,"https://ui.adsabs.harvard.edu/abs/2022AGUFM.H22J..06B/abstract, [Accessed 01-08-2023].

[10] G. Huffman, E. Stocker, D. Bolvin, E. Nelkin, and J. Tan, "GPM IMERG Early Precipitation L3 Half Hourly 0.1 degree x 0.1 degree V06," 2019, [Accessed 01-08-2023].

[11] K. Lagouvardos, V. Kotroni, A. Bezes, I. Koletsis, T. Kopania, S. Lykoudis, N. Mazarakis, K. Papagiannaki, and S. Vougioukas, "The automatic weather stations NOANN network of the National Observatory of Athens: operation and database," Geoscience Data Journal, vol. 4, no. 1, pp. 4–16, Apr. 2017.

[12] K. Lagouvardos, V. Kotroni, A. Koussis, H. Feidas, A. Buzzi, and P. Malguzzi, "The Meteorological Model BOLAM at the National Observatory of Athens: Assessment of Two-Year Operational Use," Journal of Applied Meteorology and Climatology, vol. 42, no. 11, pp. 1667–1678, Nov. 2003.

[13] A. Parmar, R. Katariya, and V. Patel, "A Review on Random Forest: An Ensemble Classifier," in International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018. Springer International Publishing, Dec. 2018, pp. 758–763.

[14] J. H. Friedman, "Greedy function approximation: A gradient boosting machine." The Annals of Statistics, vol. 29, no. 5, Oct. 2001.

[15] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," Chemometrics and Intelligent Laboratory Systems, vol. 39, no. 1, pp. 43–62, Nov. 1997.

[16] E. B. Baum, "On the capabilities of multilayer perceptrons," Journal of Complexity, vol. 4, no. 3, pp. 193–215, Sep. 1988

[17] S. H. Moon and Y. H. Kim, “An improved forecast of precipitation type using correlation-based feature selection and multinomial logistic regression,” Atmospheric Research, vol. 240, p. 104928, Aug. 2020.

[18] S. M. Lee, S. U. Han, H. Y. Won, J. C. Ha, Y. H. Lee, J. H. Lee, and J. C. Park, “A Method for the Discrimination of Precipitation Type Using Thickness and Improved Matsuo’s Scheme over South Korea,” Atmosphere, vol. 24, no. 2, pp. 151–158, Jun. 2014.

[19] W. Xiong, G. Tang, T. Wang, Z. Ma, and W. Wan, “Evaluation of IMERG and ERA5 Precipitation-Phase Partitioning on the Global Scale,” Water, vol. 14, no. 7, p. 1122, Mar. 2022.

Wind Energy Prediction Using Deep Learning Architectures

Georgios K. Floros

ABSTRACT

This thesis investigates wind energy prediction using deep learning architectures. For that purpose, we are utilizing data from Weather Research Forecasting (WRF) model and real wind-energy measurements from two distinct wind energy parks. The pre-processing process is thoroughly outlined, followed by experiments with feature extractor models such as Convolutional Neural Networks (CNN), feature extractor techniques such as the mean vector approach and the central vector approach, along with Long Short-Term Memory (LSTM), Attention and Transformer Blocks as temporal models. Results exhibit the need of efficient data pre-processing for optimal performance of models. They also show that methods containing Attention mechanism as a temporal model perform comparably, even better at some cases than LSTM. Furthermore, this study raises questions about the need of using CNN as a feature extractor, at this problem, in some cases. It also suggests that transfer learning between nearby wind energy parks is a promising approach of countering limited amounts of data and can be applied for new parks where we lack data.

Keywords: Deep Learning, Machine Learning, Wind Energy Prediction, Neural Network, Attention, Transformer Encoder, WRF

ADVISOR

Ioannis Emiris, Professor NKUA

1 INTRODUCTION

Wind energy is one of the fastest growing energy sources in the world. The demand is increasing on a yearly basis. Accurate energy prediction is important for efficient farm operations. It enables optimal energy generation, it reduces the costs because it balances demand and supply and improves the reliability of the energy supply since it helps us extract meaningful real-world patterns. Wind energy forecasting has been a concern since wind power generation was introduced. Forecasting first relied on simple wind measurements. However, overtime, more complex models have been developed that consider multiple environmental parameters such as temperature, pressure, wind speed, altitude to produce more accurate results. Persistence method, numerical weather prediction and statistical models such as ARIMA and its variations are some of commonly used for that task [1] [2] [3]. Machine learning algorithms such as SVM were even used exhibiting reliable performance [4]. Recently, thanks to the advancements of technology, neural networks, including deep learning techniques, are used to solve such tasks due to their ability to handle large amounts of data and model complex relationships between input and output variables [5] [6]. In this study we worked on wind energy forecasting using several deep learning architectures as predictive models. The task is hour-ahead energy prediction provided time-series of 6 hourly – steps. We start off with the method of a previous study [7] that helps us build on and test our methods, we consider it our foundation paper. Our input data consisted of Weather Research Forecasting (WRF) meteorological predictions and our target data consisted of wind farm real energy values. We trained separate models for each park's data, with the aim of capturing the unique patterns and characteristics of wind energy production at each location. However, we also evaluated the performance of a single predictive model for both parks, so we developed a generalized model. All in all, this study provides a complete overview of a real energy prediction problem using real life energy values. We contributed on improving the pre-processing pipeline. We carefully selected the criteria to remove noisy outliers and inaccurate or incomplete data points at all steps of this process. We also contributed on the network architecture, utilizing several state-of-the-art attention-based models which is a notable approach on such problems, and

comparing them to other established methods. In addition, we contributed by providing problem insights through the interpretation of results, leading to useful conclusions and suggestions such as the need for a generic multi-park all-in-one model.

2 BACKGROUND & RELATED WORK

Artificial intelligence has taken centre stage in computer science thanks to constant advancements in technology and the creation of fast and powerful computational systems. Neural networks are a crucial tool in this field as they process large amounts of data and solve complex problems. Deep learning, a subset of artificial intelligence, utilizes neural network models deploying them into deep architectures.

Models considered state of the art at performing certain tasks include *Convolutional Neural Networks (CNN)* [8] mainly concerned with face recognition, natural language processing (NLP) applications, optical character recognition (OCR) and image classification. *Long – Short term memory networks*, [9] a special type of recurrent neural networks (RNN), capable of learning "long - term dependencies." They are indented to retain knowledge over time and handle sequences of data. *Attention mechanism* [10] is also a technique used in deep learning networks usually applied on LSTM encoder – decoder architectures. Its purpose is to emphasise the most relevant parts of the input sequence in a flexible manner. It uses three vectors, key and value describes each state of the encoder and query typically represents the last hidden state of the decoder. Attention is described as defined by the following equations:

$$C = \sum_{j=1}^{n} a_i \, v_j$$

Where c represents the context vector for an input sequence $X = (x_1, x_2, \ldots, x_n)$ of length The weight is computed by:

$$a_j = \frac{\exp\left(e_{q_j, k_j}\right)}{\sum_{i=1}^{n} \exp\left(e_{q_i, k_i}\right)}, \qquad e(q,k) = \frac{(q)^T k}{\sqrt{d_k}}$$

Where $e(q,k)$ is the alignment score function, researchers have proposed dot product attention while a more recent work [11] proposed it scaled by $\frac{1}{\sqrt{d_k}}$, where d_k is the dimension of key vector.

Self – Attention [11] is a method by which we apply the attention mechanism to each position of the same sequence. It is described by the following equation:

$$\text{Attention}(Q,K,V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where Q, K and V are tables that accumulate the values of query, key, and value vectors.

Positional embeddings are also a method used since attention mechanism does not include positional information. Positional embeddings are added to preserve information about the order of timesteps. They can be both fixed and learned. They are usually aggregated and added to the input data values. Fixed positional embeddings proposed on bibliography [11] are described by:

$$\text{PE}(pos, 2i) = \sin\left(\frac{pos}{n^{2i\,/\,dmodel}}\right), \qquad \text{PE}(pos, 2i+1) = \cos\left(\frac{pos}{n^{2i\,/\,dmodel}}\right)$$

where pos defines the position in the input sequence, i is the dimension and n refer to a user defined variable (recommended value is 10.000). Learned positional embeddings learn a mapping function through the training process [12].

As for wind energy farms, wind energy production is strongly associated with wind speed. The produced energy increases as wind speed increases. The design and operation of a wind turbine requires a certain wind speed range. The limits of this range are the cut-in speed and the cut-out speed. Cut in speed is when the turbine starts spinning for the first time and generates power. When wind speed increases gradually, it surpasses cut-in speed and reaches the maximum limit turbine design can support. To avoid the risk of damage on the rotor of the

turbine, cut-out speed is activated, which stops the operation with the help of suitable auditors [13].

3 METHOD

3.1 Data Pre-processing

We work with data from two different parks. At first, we discuss the methods we used to prepare the target data pre-processing.

The *target data* we use, produced from a SCADA system, consists of real energy measurements from two different wind energy parks located in central Greece. Park A consists of 16 turbines with a total of 12,000 kW total capacity while park B consists of 17 turbines with a total of 10,800 kW total capacity. These data faced availability issues. Turbines may have not performed at one hundred percent for a period due to maintenance or economy reasons. We need to exclude low availability points.

We are provided with real wind speed measurements for 10-minute intervals and energy measurements for 15-minute intervals. We adjust them to 1-hour time intervals. Then we plot the energy – wind speed diagram (Figure 3.1) where data points form a dense curve. To filter out the noise we use several criteria. We use line segments where we consider the points outside the curve as outliers and possible low availability points. We set a minimum energy value of 400 kW since it is possible for turbine blade to be slightly displaced by the wind when park is closed and record such low values. We also exclude data points with wind speed less than 5 m/s and 25 m/s since they are measured outside of cut-in and cut-off speeds. The data curves before and after pre-processing are displayed in Figure 3.1 and Figure 3.2.

The input data used in this study consisted of 18 WRF predicted features: X and Y components of wind, temperature, and pressure, all at 10, 80 ,100, 120 meters height, surface pressure, snow water equivalent and daily total snow and ice.

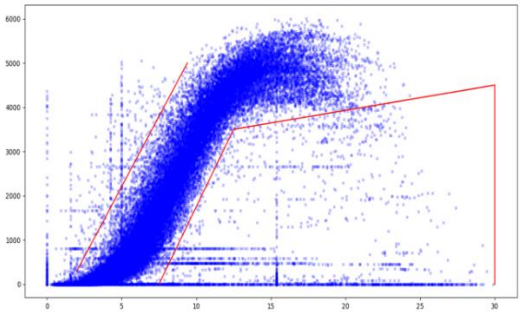


Figure 3.1: Power - Wind plot before pre-processing

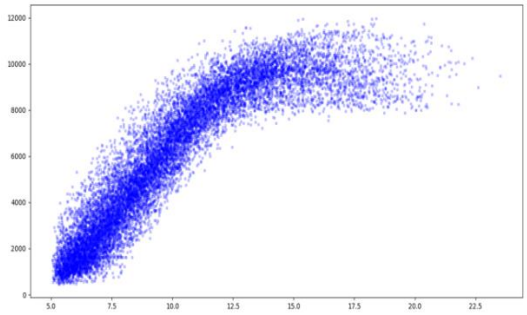


Figure 3.2: Power - Wind speed final plot

It was extracted from parameterized WRF models executed on a rectangular region covering a large fraction of mainland Greece, provided by the work of our foundation paper [7] [14]. The model produces a WRF grid that records weather or wind patterns. We get forecasts at every location of the grid with a 1 km resolution. Using the specified coordinates (longitude and latitude) of the wind parks we obtain the grid for the area of interest from the initial WRF. For each point of this grid the 18 WRF predicted features were generated for an hour interval. Park A has a 7 x 8 grid, while park B has a 6 x 8 grid.

In the pre-processing phase the input data was matched with the target data using time criteria. For each target label corresponding to a timestamp of t hours we saved the WRF predictions for the hours $t, t-1, \ldots, t-5$. As a result, each input element forms the following vector shape:

timesteps × park grid height × park grid width × wrf features

3.2 Models and components

Our architecture overview consists of two main components. A feature extractor, to identify geographical trends and a temporal model, to capture temporal patterns. An architectural overview is displayed in Figure 3.3. The model's output is described by the following relationship:

$$y = h \circ g \circ f(x)$$

Where f represents the feature extractor, g represents the temporal model and h is the output MLP.

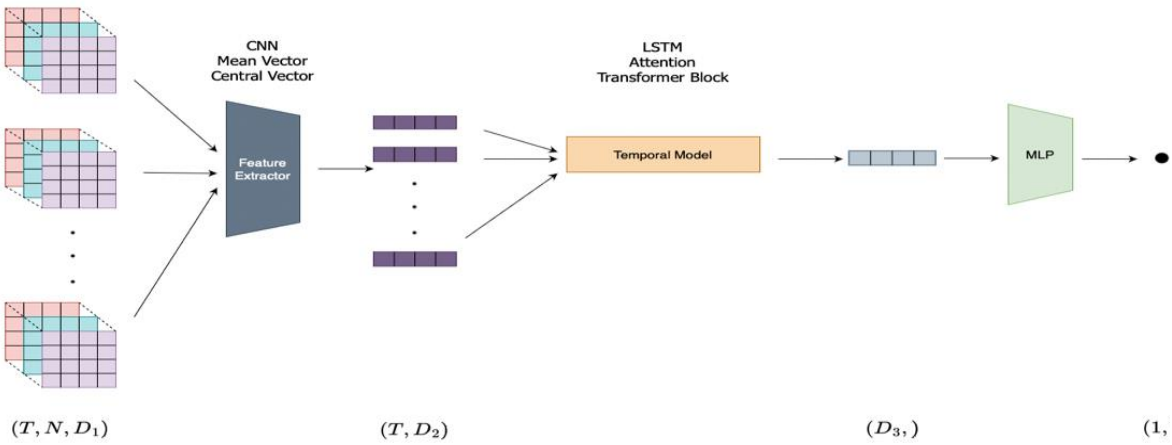


Figure 3.3: Architectural overview

3.3 Feature extractors

As feature extractors we use several methods. *Convolutional Neural Networks* (CNN), which is defined as:

$$f : \mathbb{R}^{H \times W \times D} \longmapsto \mathbb{R}^{D'}$$

where H and W represent the WRF grid's height and width, and D represents the input features dimension. H and W are set to 7 and 8 for park A and 6 and 8 for park B.D represents the input features number that is equal to 18 and D' represents output features dimension that is equal to 64.

Another method we deploy is the *mean vector* approach. We utilize the mean 18 WRF features of all the points of the WRF grid instead of using all the feature vectors in the grid.

$$\vec{v} = \frac{1}{H \cdot W} \sum_{i}^{H} \sum_{j}^{W} v_{i,j}$$

Where H and W represent height and width. This results in an output shape of 18 features for each of our 6 time-steps.

Similarly with the mean vector we keep the central 18 WRF features of all the points of the WRF grid for each time step. To determine the *central point*, we use the following equation:

$$\overrightarrow{y_i} = (lat_i , long_i)$$

$$\vec{\mu} = \left(\frac{1}{N} \sum_{k=0}^{N} lat_k , \frac{1}{N} \sum_{k=0}^{N} long_k \right)$$

$$L_1(\overrightarrow{y_i} , \vec{\mu}) = \sum_{j=0}^{2} \left| y_i^j - \mu^j \right|$$

$$c(\overrightarrow{y_i}, \vec{\mu}) = \arg_i \min \mathrm{L}_1 (\overrightarrow{y_i}, \vec{\mu})$$

Where N represents the number of wind turbines, μ represents their mean coordinates, $y_i$represents the coordinates of each WRF points and c represents the index of the central vector. In case of mean and central vector method a multi-layer perceptron (MLP) with one hidden layer and output size 64 with a ReLU activation function follows. The MLP is described as followed:

$$f : \mathbb{R}^D \longmapsto \mathbb{R}^{D'}$$

where D presents the input's number of features and D' the output features number. The purpose of using MLP is to feed our temporal model with extra features. The MLP in our method receives 18 input features and outputs 64.

3.4 Temporal Models

After extracting the output from feature extractor model, we feed it to our temporal model. We use LSTMs. *LSTM* is defined as:

$$g : \mathbb{R}^{T \times D} \longmapsto \mathbb{R}^{D'}$$

where T represents the number of time steps, D the number of input features and D' the number of output features. In our method we input 64 features, we use an LSTM with a hidden size of 256 nodes followed by a dropout layer of 0.2 dropout rate. That results in an output shape of 256 features. We also experiment with bi-directional LSTMs where the outputs are concatenated.

We also experiment with *Self-Attention* based on the paper [11]. In our method we use as key and value dimension the value 64 the same as the features input dimension. Our model is defined as:

$$g: \mathbb{R}^{T\times D} \longmapsto \mathbb{R}^{T\times D'}$$

where T represents the number of time steps, D the number of input features and D' the number of output features. The model's output is computed and added back on the input. We flatten the output and end up with 384 final features.

To create a *Transformer Block* we extend self-attention adding an MLP with one hidden layer and output size 64 and ReLU activation function. Transformer Blocks have the same input and output shape, so we flatten input and end up with 384 output features as a final shape.

3.5 Extra Components

Learnable and fixed positional encodings used in other works [11] [15] are used to help attention and transformer blocks receive positional information. We receive input information, incorporate positional information of the same shape, and then forward the output on attention or transformer blocks.

$$w = PE(t) + x, \qquad PE: \mathbb{R} \to \mathbb{R}^D, \quad w: \mathbb{R}^{N\times D} \to \mathbb{R}^{N\times D},$$

x denotes the input data and w represents the final input data after adding positional encodings.

We use Adam as our optimiser, with learning rate 0.001 and batch size is set to 48. Regarding the validation and test sets we use during training; they are a part of the data set that consist of data equally distributed through seasonality. That is because we want our network to predict accurate values for all seasons without biases. On both parks we select data from the same exact months for validation and test set. We have no overlaps. As a loss function during training, we slightly modify *MAE* so that y_i is in the same range as y_p. To evaluate the test set, *Mean Normalized Absolute Error (MNAE)* is used:

$$\text{MAE}_{\text{custom}} = \sum_{i=1}^{n_{\text{samples}}} \left| \frac{y_i - \min_y}{\max_y} - y_p \right|, \quad \text{MNAE} = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}} \frac{|y_i - y_p|}{c}$$

y_i is the actual energy value and y_p is the predicted value and c is a constant value set to 12.000 in our experiments.

4 RESULTS & DISCUSSION

First, we test the results of applying our pre-processing method. We compare with the previous method of pre-processing. We use min-max normalization and CNN + LSTM model.

Data	**MNAE**	**Data**	**MNAE**
Old preprocessing	14.1656 ± 0.3885	Old preprocessing	13.3116 ± 0.4404
Ours	12.3939 ± 0.2880	Ours	13.0028 ± 0.6116
Table 4.1: Park A		Table 4.2: Park B	

Significant changes were noticed with the old data compared to the new in both parks. In particular, the metric improved on both parks, especially in park A (12.3939 compared to 14.1656), but also in park B (13.0028 compared to 13.3116). These results highlight the importance of data preprocessing as a critical step in the development of accurate and reliable neural network models.

Second, after some fine – tuning experiments with several normalization and output functions we decide we will be using *Z-score* normalization and *sigmoid* output activation function as it seems like that configuration fits the distribution of our data better. Third, we start experimenting with different feature extractors.

Feature extractor	**MNAE**	**Feature extractor**	**MNAE**
CNN	11.7516 ± 0.3657	CNN	12.6021 ± 0.4108
Mean vector	12.1674 ± 0.2057	Mean vector	12.4049 ± 0.3539
Central vector	12.0207 ± 0.2573	Central vector	12.1578 ± 0.0937

Table 4.3: Park A	Table 4.4: Park B

For park A, we notice central vector performs a bit better than mean vector method (12.0207 compared to 12.1674) while CNN gives us the best metric (11.7516). Despite that, the results of CNN do not have big differences with the other methods. Potentially because the input WRF values are close from one point of the grid to another and the spatial patterns the CNN is designed to capture are not that significant. For park B, central vector outperformed the other methods (12.1578). CNN in that case did not help our model as expected. An explanation is, the geographical distance between some points of the grid and the location of the turbines created instability. This instability could be because the CNN may have difficulty capturing the spatial relationships between these distant points. The results of the experiments raise questions about the essentiality of CNNs for our problem, as a simpler model achieved similar or even better performance.

Fourth, we are experimenting with different temporal models and positional encodings, we are using CNN as feature extractor, Sigmoid output activation with z-score normalization and custom mae as a loss function.

Temporal Model	**Pos. encodings**	**MNAE park A**	**MNAE park B**
LSTM	-	11.7516 ± 0.3657	12.6021 ± 0.4108
Bi-LSTM	-	12.4621 ± 0.5107	12.4230 ± 0.3290
Attention	-	12.2100 ± 0.5640	13.0851 ± 0.3866
Attention	Learnable	11.9796 ± 0.3330	13.0798 ± 0.4014
Attention	Fixed	12.0599 ± 0.3525	12.8692 ± 0.2602
Transformer Block	-	12.2671 ± 0.4067	12.9023 ± 0.6318
Transformer Block	Learnable	11.8685 ± 0.2706	12.5219 ± 0.4044
Transformer Block	Fixed	11.6368 ± 0.2032	12.9150 ± 0.5425
2 Transformer Block	-	11.9848 ± 0.2606	13.1800 ± 0.6846
2 Transformer Block	Learnable	11.9009 ± 0.1874	12.6517 ± 0.3731
2 Transformer Block	Fixed	12.0535 ± 0.3532	12.7696 ± 0.4674
Table 4.5: Temporal model and pos. encodings experiments for both parks			

For park A, the best performance was achieved using a Transformer block with fixed positional encodings as it outperforms the LSTM (11.6368 compared to 11.7516). We observe that two stacked transformer blocks compared to one slightly improve the metrics (11.9848 compared to 12.2671). Positional encodings tend to improve results in both attention and transformer block components. Learnable positional encodings help attention (11.9796 compared to 12.2100), and fixed positional encodings help single transformer block significantly (11.638 compared to 12.2671). However, major improvement is not observed when combining them with two stacked transformer blocks. Learnable slightly help the models (11.9009 compared to 11.9848) while fixed produce slightly worse results (12.0535 compared to 11.9009). Similar results are obtained for park B. The best metric though is obtained by a bidirectional LSTM (12.4230). Combining information from both time directions proved useful for park B, differences are not significant though.

Overall, LSTM architectures and attention-based architectures showed comparable performance when used to process and analyse our time-series data. That suggests that both architectures can be considered as viable options as temporal models in our task. As for positional encodings, our models improved with their addition in most of the cases. Comparing learnable and fixed positional encodings, we do not observe major differences between them. Learnable position encodings thought improved the model in all cases they were used.

Lastly since parks A and B are 6 km apart, and meteorological WRF predictions should be similar for both parks, we combine park data and create an all-in-one model. We use the standard CNN + LSTM model.

	MNAE park A	**MNAE park B**
Single, A	11.7516 ± 0.3657	-
Single, B	-	12.6021 ± 0.4108
All-in-one	11.9604 ± 0.3097	11.9030 ± 0.3462
Table 4.6: All-in-one model comparison		

The findings of this experiment are really promising. The metric of park A gets a bit worse (11.9604 compared to 11.7516). This has to do with the fact we have

different max values on park A and park B. Park A values are capped on 12 kWh and park B values are capped on 10.2 kWh. As a result, the all-in-one model misses some high values for park A. However, park B benefits a lot (11.9030 compared to 12.6021) from this merge. That leads us to think that the concept of merging the park train data might work if we encode the information of which park data come from for our model. This experiment provides us with some interesting views. It shows that transfer learning between two nearby parks is a viable option.

A visualisation of the test set predictions can be seen bellow. In Figure 4.1 we have included a visual representation of park A best model's (CNN + transformer block + fixed pos.encodings) predicted values compared to the actual ground truths. We have also included in Figure 4.2 a visual representation of the predictions of the model trained with old pre-processing data (Table 4.1). We can observe that especially low energy values, but also high energy values are modelled better with our best model.

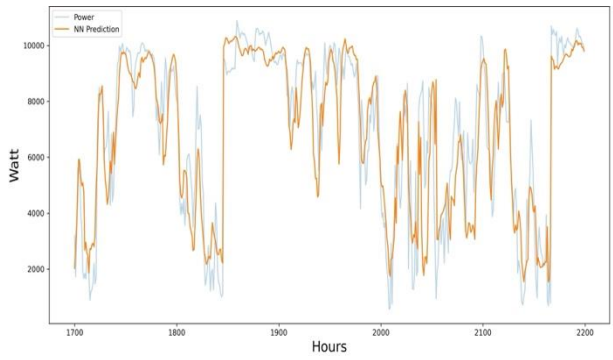


Figure 4.1: Ground truth – Prediction, park A

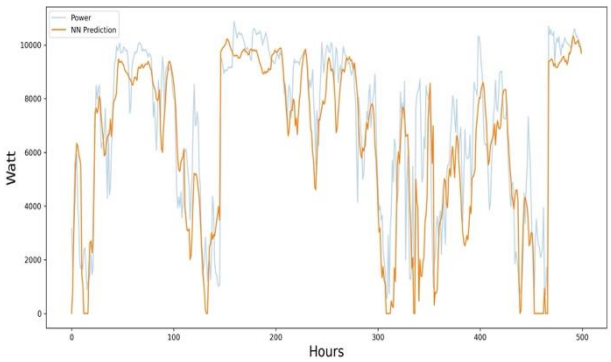


Figure 4.2: Ground truth–Prediction, park A old

5 CONCLUSIONS & FUTURE WORK

The first part of our project involved developing a data pre-processing method based on our foundation paper [7]. Testing was performed in comparison with this paper. The results showed that our proposed method exhibited better performance on the same exact test set. These findings indicate the crucial role

of efficient pre-processing in such tasks as it made a great impact on the performance of our models.

In the second part we implemented the model architecture. Our model architecture consisted of a feature extractor and a temporal model. As feature extractors we implemented the CNN, mean vector, and central vector approaches. Our results were similar in all cases as in the first park CNN performed quite better while in the second park the central vector approach outperformed the rest. That raises questions about the essentiality of CNNs for our problem as our experiments showed that a simpler approach achieved similar or even better performance. For temporal models, we compared LSTM, attention, and transformer blocks with the addition of positional encoding. Our results showed that attention-based perform comparably and even better than LSTM architectures as temporal models. In addition, while no clear preference emerged between learnable and fixed positional encodings, their addition to attention-based models was beneficial.

Finally, we built an all-in-one model using data from both wind farms. By leveraging data from multiple sources, we achieved partial improvement in forecasting accuracy. Certain limitations this study faced is the lack of data, the data provided were valuable, but a larger dataset would have provided more robust and reliable results. In addition, is worth noting that the data provided required denoising. While our method addressed this issue, it remains a challenge that requires further exploration. In addition, although the WRF provided valuable meteorological features for our input data, it is a forecasting model that contains errors. An improvement could be to incorporate real meteorological data to model the WRF's historical prediction errors and potentially account for the differences between the predicted and actual meteorological values. Additional improvements could involve leveraging aerial images of the wind farm to further encode the park's location and geography. This extra signal can provide valuable information about nearby terrain features and improve the model's ability to make accurate predictions. Another approach could be the use of unsupervised pre-training tasks (self-supervised learning) [16]. By training the model on WRF data from several areas in Greece for a pre-

training task, the model can learn common patterns and features in the data that are relevant to wind energy prediction.

REFERENCES

[1] Milligan, M. R., et al. (2003). Statistical wind power forecasting models: Results for U.S. wind farms: Preprint.

[2] Almoataz Youssef, A. (2012). Short term wind power forecasting using autoregressive integrated moving average modelling. Proceedings of the 15th International Middle East Power Systems Conference.

[3] Kavasseri, R. G., & Seetharaman, K. (2009). Day-ahead wind speed forecasting using f-ARIMA models. Renewable Energy, 34(5), 1388–1393.

[4] Mohandes, M. A., et al. (2004). Support vector machines for wind speed prediction. Renewable Energy, 29(6), 939–947.

[5] Li, G., & Shi, J. (2010). On comparing three artificial neural networks for wind speed forecasting. Applied Energy, 87(7), 2313–2320.

[6] Moustris, K. P., et al. (2016). 24-h ahead wind speed prediction for the optimum operation of hybrid power stations with the use of artificial neural networks. In Perspectives on Atmospheric Sciences (pp. 409–414).

[7] Christoforou, E., Emiris, et al. (2021). Spatio-temporal deep learning for day-ahead wind speed forecasting relying on WRF predictions. Energy Systems, 14(2), 473–493.

[8] O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks.

[9] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.

[10] Bahdanau, D., et al. (2014). Neural machine translation by jointly learning to align and translate

[11] Vaswani, A., et al. (2017). Attention is all you need.

[12] Gehring, J., et al. (2017). Convolutional sequence to sequence learning.

[13] Cole, S. (2021). TheRoundup. https://theroundup.org/wind-turbine-power-curve/

[14] Research team of A. Anastasiou and I. Z. Emiris “WRF model parameterization and adaptation of mainland Greece, with final feature extraction”

[15] Devlin, J., et al. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding.

[16] Erhan, D., et al. Why does unsupervised pre-training help deep learning? Journal of Machine Learning Research,

Analysis of AlphaFold 2 Algorithm

Vasiliki L. Pitsilou

ABSTRACT

AlphaFold 2 [1] has been a revolutionary algorithm in the field of computational biology. It made a huge leap in predicting the three-dimensional structure of a protein. Predicting the structure of a protein is a problem that has been challenging scientists for decades, and its solution is to this day a very desired one, as protein structures play a crucial role in understanding the function of proteins. And subsequently, understanding the function of proteins plays a crucial role in understanding the functions of the human body and its diseases.

AlphaFold 2, an AI-model built by Google DeepMind [2], is able to accurately predict protein structures for a big number of proteins, far outperforming all other methods developed before it. But how did it manage such an impressive performance in such a difficult problem? This thesis presents the AlphaFold 2 algorithm after giving some context on proteins and the protein folding problem. It also includes an experimental part, where the algorithm is executed and some technical details are examined.

Keywords: protein folding prediction, AlphaFold 2 algorithm, protein folding problem, machine learning, deep learning

ADVISORS

Ioannis Emiris, Professor NKUA
Andreas Zamanos, MSc
Georgios-Alexis Ioannakis, PhD

1 INTRODUCTION

Proteins are macromolecules that take part in nearly every process in our cells. They are constructed inside the cell and perform a large number of different tasks like catalyzing chemical reactions (e.g. digestion), transporting nutritional components throughout our bodies, attacking foreign objects entering our bodies (antibodies), building cell and tissue structures, regulating processes in our bodies (protein hormones) and others.

The building blocks of proteins are smaller molecules called amino acids. To form a protein, amino acids attach to each other forming a chain, which then folds to shape a 3-D structure (Figure 1). This structure allows a protein to perform the task it is responsible for.

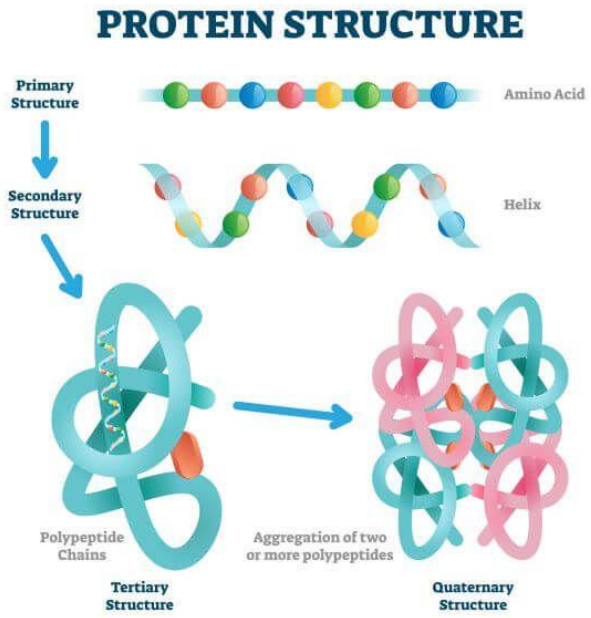


Figure 1

The proper folding of proteins into their native state is essential for their normal function. However, various factors can disrupt this process, leading to protein misfolding. Mutations in the genetic code, as well as environmental factors such as temperature or the presence of certain chemicals, can interfere with the folding process. When a protein misfolds, it means that it adopts an incorrect

three-dimensional structure compared to its native state. This misfolded protein may become unstable, lose its functional properties and be unable to carry out its intended task effectively. Misfolding can disrupt the protein's ability to interact with other molecules, impair its enzymatic activity, or interfere with its structural role. Since proteins play crucial roles in numerous biological processes, the consequences of protein misfolding can be severe.

Knowing a protein's structure is very important for medical and biological research. The structure of nearly 100,000 proteins (compared to the billions of known sequences) has been found experimentally [3]. But determining a protein structure experimentally is not an easy task: it is expensive and time consuming, taking months to years to complete. The scientists' goal is to be able to determine the structure of a protein in the three-dimensional space, given its amino acid sequence (Figure 2). This is theoretically possible (Anfinsen, 1972) and if that is achieved, scientists will then be able to determine the structure of every protein. This is the so-called protein folding problem and what AlphaFold 2 aims to solve by using trained deep neural networks, especially Transformers [11], and by deploying the **self-attention** technique. Self-attention is the technique of making the network focus only on some parts of the input that seem more useful than other input parts in predicting the output. That way, the network extracts more meaningful information from the input and outputs a highly accurate prediction.

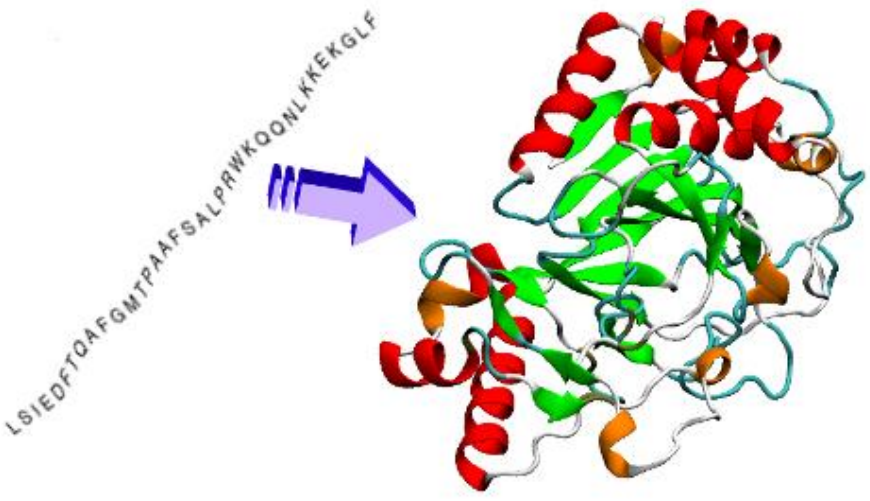


Figure 2

2 THE ALPHAFOLD 2 ALGORITHM

AlphaFold 2 takes a protein amino acid sequence as input and outputs the most probable structure of that sequence. The program's output is a structure for the protein in question along with confidence scores (ranging from 0 to 100%) for each amino acid. The program is split into two big parts: the feature embedding and Evoformer (first part) and the Structure module (second part). In the first part, information from other proteins, related to the input one, is highly processed and used to build some structures. These structures will then be the input to the second part, which will use them to end up in a protein structure. The network uses recycling: the outputs of an iteration are used to update the input to the next one. The number of iterations is defined in the program.

AlphaFold 2 widely uses neural networks. To sufficiently train those networks, AlphaFold 2 relies on a large amount of protein data. These data can be found by searching protein databases. The proteins searched are evolutionarily related to the protein in question, meaning they also share common amino acids. Using information from proteins with known structures, the model can see how evolutionarily related proteins fold and that way extract information about the structure in question. Using information from proteins with unknown structures, it extracts information by aligning their sequences and finding patterns which reveal the proximity of two amino acids in the final protein shape. The last information is embodied in a table called MSA table.

During inference, AlphaFold 2 receives input features from the input protein sequence, the MSA table and templates (proteins with similar structures to the one we want to predict), and outputs atom coordinates and confidence score per amino acid.

The first part of the algorithm (Figure 3) builds an MSA and pair representation out of the MSA table, the input sequence and the templates. The MSA representation encodes information of evolutionarily related proteins, while the pair representation about the correlation between each pair of amino acids in the input sequence. During that part, there is a high information exchange between pair and MSA representation, and self-attention methods are used.

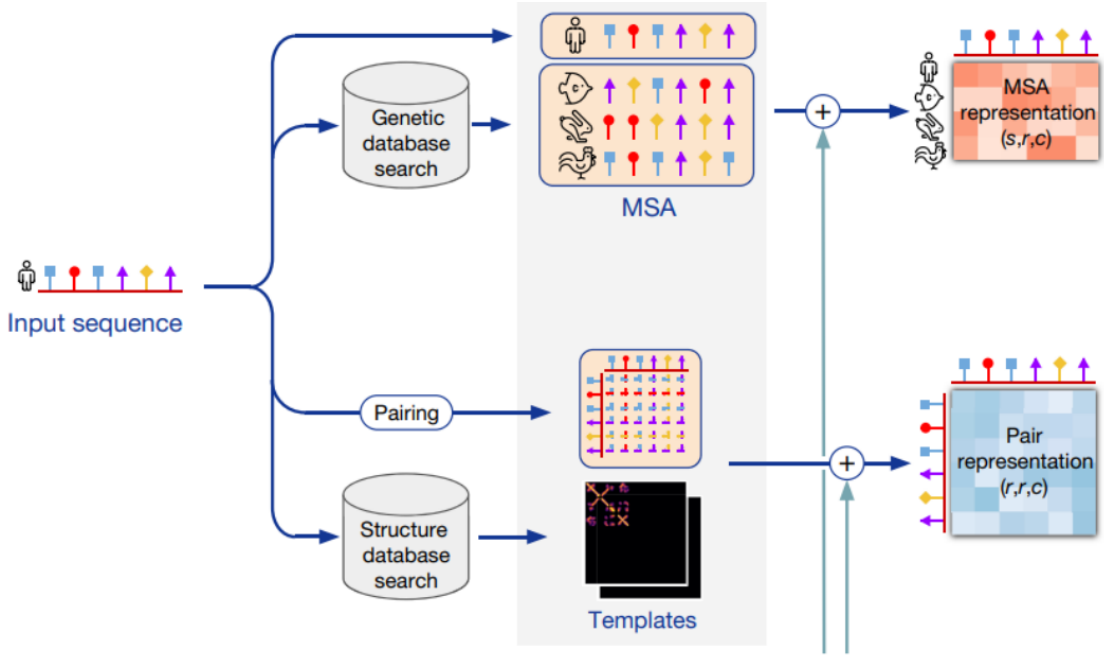


Figure 3

Then, having acquired the final form of the MSA and pair representations, we use them to build a 3-D protein structure out of them. This is done in the second part of the algorithm, or the Structure module

The structure module (Figure 4) takes two of the algorithm first part's outputs as input: the pair representation and the single representation. The single representation is the MSA table's first row and represents the input amino acid sequence. A 3-D representation of the protein's structure is initialized in the algorithm. The module consists of 8 layers and in each layer, the single representation and the 3-D representation are updated. The processed version of the 3-D representation will be the output of the entire algorithm. Updating the single representation is needed because this representation will give us information on the angles of the 3-D structure.

In the Structure module, the single representation is updated after self-attention is performed on it and after receiving bias from the pair representation. The processed single representation is then used, along with the unprocessed one, to predict torsion angles (angles between planes formed by atoms of one or two amino acids), which are subsequently used to predict the final atom coordinates.

For the prediction of confidence, the processed single representation is given as input to a trained network, which returns the confidence per amino acid.

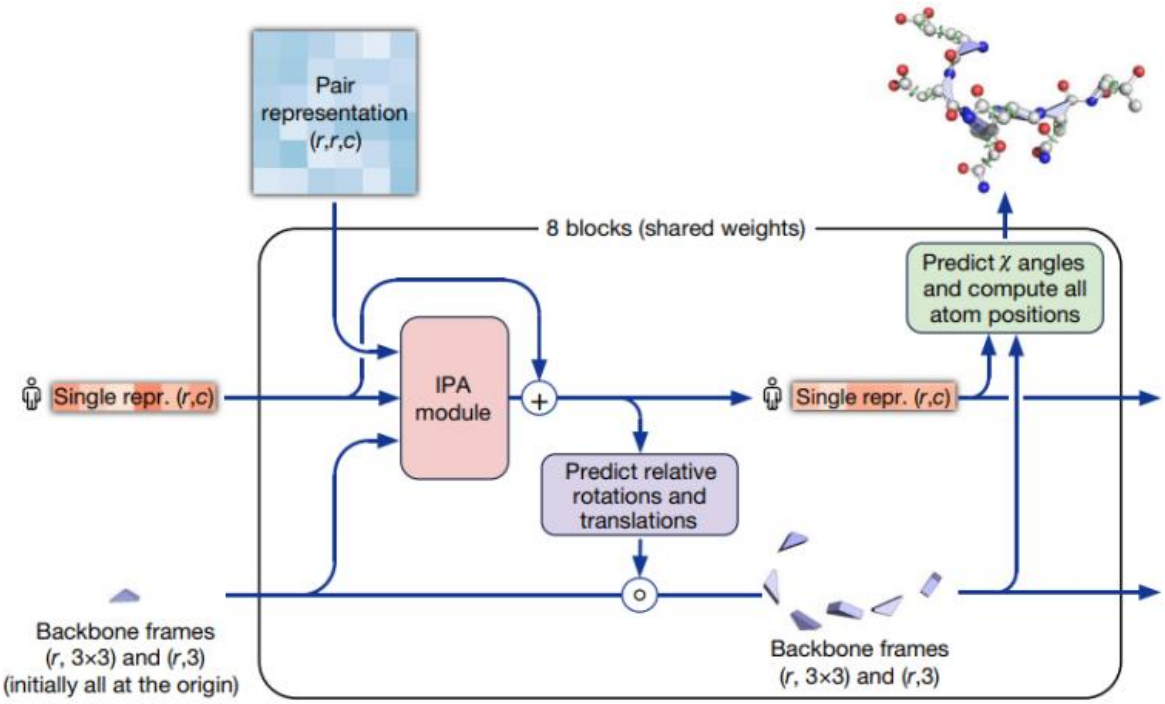


Figure 4

Returned by the structure module are the atom coordinates and the confidence measure. Lastly, relaxation is performed on the predicted structure. When a molecular structure is predicted by a computer program, it may contain structural violations, meaning violations that do not appear in natural structures. So the structure undergoes a procedure called relaxation in order to resolve as many violations as possible. Both the relaxed and unrelaxed versions of the protein structure are returned by AlphaFold 2. The structure module's outputs (coordinates and confidence measure) will be organized in a PDB file. A PDB file [4] is a file that represents a molecule structure, by assigning coordinates to all its atoms.

3 RUNNING THE ALGORITHM

AlphaFold 2 is an open-source software available on GitHub [5], with detailed instructions on how to run it. AlphaFold 2 can only be run in inference mode and is not available for training or fine-tuning. Running it requires a Linux-running machine, around 3 TB of storage space for the full download of protein databases and a powerful GPU. For systems with less available memory, there is also the option of downloading a reduced version of databases (a database of 17GB will be downloaded instead of the full one of 1.8 TB, still leaving the total required size to around 1.2 TB). AlphaFold 2 needs a significant amount of time to set up: many hours (even a day or more) are required to download the data needed for running the algorithm.

OpenFold [6] is a faithful reproduction of AlphaFold 2 implemented in PyTorch, whereas AlphaFold 2 was developed for JAX workflow. OpenFold does give the option for training or fine-tuning as opposed to AlphaFold 2, and can be faster in inference mode. Still, its setup requires the same resources mentioned for AlphaFold 2 (multiple hours of downloading databases as well as storage). In an attempt to run the AlphaFold 2 algorithm, we chose to run OpenFold, since it would allow us to also experiment with some fine-tuning. We also use Docker. Docker [7] is a software that allows a user to perform any installations and runs in a secluded environment, called a container. That way, the program will not have to interfere with the whole system where the program will be run and allows independence, especially in the case of systems used by many people. Docker is a simple way of running OpenFold.

The output of the program is a PDB file. A PDB file can then be used in a visualizing software like Chimera [8]. Such software visualizes a structure as encoded in a PDB file. The PDB file returned by OpenFold for the MCHU Calmodulin protein can be visualized into the following 3-D structure, using Chimera 1.16 (Figure 5):

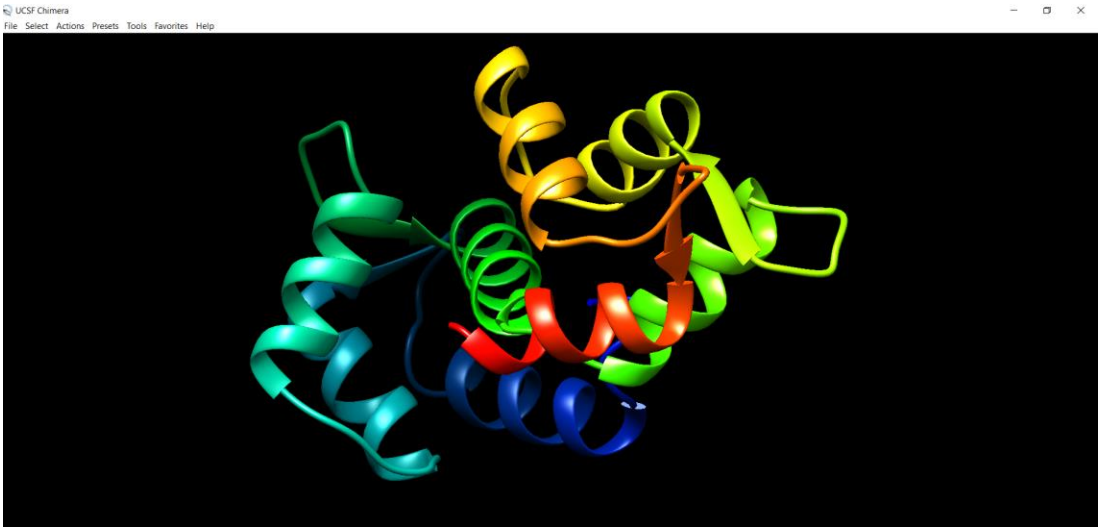


Figure 5

OpenFold offers the possibility of running the program in Google Colab (Figure 6) [9]. Google Colab is an environment where Jupyter notebooks can be run without the use of setting up GPU or CPU, since they are provided by Google. This is a pretty simple way of running OpenFold without having to download and run it manually. To run OpenFold on Google Colab, simply enter the amino acid sequence on the first cell, choose the model you want to use (OpenFold and AlphaFold 2 have slight differences) and then run all the cells. This program downloads all the databases on the Google cloud environment. The running time for an amino acid sequence of 70 amino acids was 40 minutes, including all downloads of databases. After being completed, the program downloads a zip file containing the PDB file and the predicted error in a JSON file. It also visualizes the 3D structure along with its confidence prediction. In the 3D structure, the coloring denotes the confidence areas:

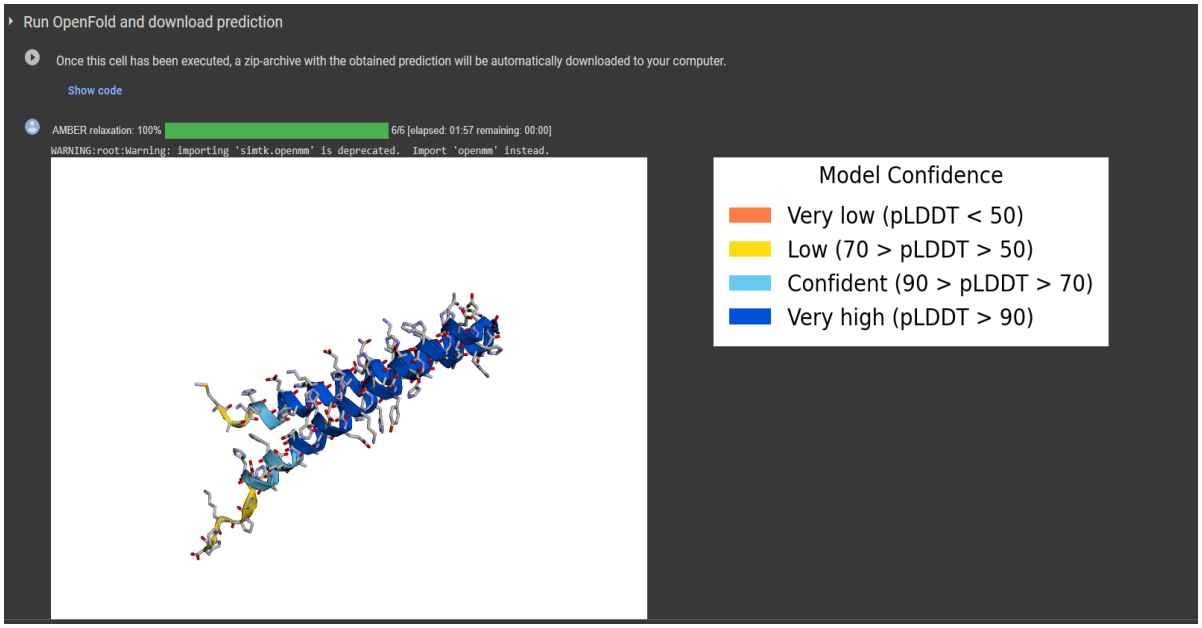


Figure 6

Lastly, AlphaFold database [12] contains over 214 million protein structures and their confidence metrics, as predicted by AlphaFold 2.

4 RESULTS

We saw how AlphaFold 2 can be run using OpenFold. All we used was a simple computer system running Linux and an 8 GB GPU. Even though we needed a large storage space of around 3 TB (which can also be reduced to 1.2 TB as mentioned in Chapter 3), it is still impressive that anyone can obtain a protein structure estimation, corresponding to an amino acid sequence of their choice within minutes, simply by running a program on their computer or on Google Colab. Until some years ago, a protein structure could only be obtained experimentally by scientists, after months of effort. This demonstrates the power of computer science, computer systems and machine learning. In the previous chapter, predictions were obtained using trained models. But taking a step further, what would happen if someone wanted to fine-tune the model, or even train it from scratch, needing even higher resources? AlphaFold 2 does not offer the option of training or fine-tuning, probably because they require resources

available only within scientific communities. But OpenFold remains an option for fine-tuning or training the model. In order to fine-tune the model on a selected group of proteins, those proteins first need to be aligned. Alignment will be used for the construction of the aforementioned MSA (Multiple Sequence Alignment) table. In an alignment procedure, all proteins are aligned based on a model protein, in order to find common amino acids with it. Since all proteins have to take the role of the model protein (all proteins have to be compared with all others), the alignment procedure leads to a high complexity

The problem is that this step is not that simple. After trying to align a set of 900 proteins using both alignment scripts, the result was the following

- For the former, alignment of approximately one protein per day (which would result in 900 days of aligning).
- For the latter, usage of 90% of computer memory on average. This amount of mmory consumption prevents others from using the same computer machine.

Alignment running for 3 days consecutively, was killed manually afterwards. This method is faster than the previous one. These data show us that it is infeasible to run the alignment script in a short amount of time, but one can overcome this obstacle by having precomputed alignments available. In that case, alignment of proteins is not needed. Besides alignment, we also expect fine-tuning to take a long time (at least a few days). The amount of time needed for aligning prevented us from fine-tuning the model. Fine-tuning is a much harder task, although not infeasible, given that one can invest time and effort in it. But, if alignment could become more efficient, many scientific teams or individuals would experiment with training and/or fine-tuning, leading to a better understanding of the algorithm and even improvement of it.

At this point, we could not expect the same thing for training or fine-tuning, since solving such a problem using machine learning is, by its nature, resource-intensive and training this model using such a large amount of data could not be any faster. AlphaFold 2, even with its weaknesses in terms of resources or time needed to train or fine-tune it, is still a very sophisticated algorithm, which using state-of-the-art techniques has managed to solve a very important biological

problem, at least to a certain extent. But as science, by its nature, aims to solve problems (even after having just solved a very important one), scientists will try to improve the algorithm's efficiency and more importantly, obtain even more accurate protein structures.

5 CONCLUSIONS AND FUTURE WORK

AlphaFold 2 has been able to predict protein structures given their amino acid sequence, with a high accuracy for the majority of proteins. The algorithm has been considered revolutionary and is indeed remarkable, since it has achieved significant advancements in predicting protein structures and thus managed to solve the protein folding problem to a large degree. That way, it expanded our knowledge on previously unknown protein structures and the molecular world and showed how machine learning algorithms can be reliable and powerful tools for solving biological problems. It has also given the scientific world new tools and ideas on how to tackle specific problems using machine learning techniques, opening up new possibilities for scientific research.

However, we should keep in mind that machine learning algorithms make predictions based on data they have been trained on. So one should be careful when using a predicted protein structure and not consider it as ground truth, but also take into consideration the confidence scores provided with every structure. Those confidence scores indicate, as mentioned, the model's certainty for the prediction of each amino acid's position in the 3-D structure. The scientific community will strive for even bigger breakthroughs in the field of structural biology. The AlphaFold 2 algorithm has been a precious tool in the hands of scientists, which will continue playing a vital role in advancing our knowledge on proteins and facilitating various research endeavors.

What we think for a future project is testing the algorithm's generalization, using the class of membrane proteins. Membrane proteins [10] are proteins existing in or interacting with biological membranes. The generalization test can be done by fine-tuning OpenFold on the class of membrane proteins and then testing its

performance on membrane protein structures. Then test the performance of the non-tuned model on the same dataset. Finally, compare the performances of the two programs. If the fine-tuned version does not perform much better than the non-tuned one, we would conclude that AlphaFold 2 can perform well on predicting protein structures it has not been specifically fine-tuned on. Ideally, we would expect similar performances for both models, which would mean that AlphaFold 2 performs well for all protein classes, without the need for fine-tuning. For this project, we will again confront the alignment problem discussed in the previous chapter, but we will invest a significant amount of time both in protein alignment and fine-tuning of the model, hoping to overcome it.

REFERENCES

[1] Alphafold. https://www.deepmind.com/research/highlighted-research/alphafold. Accessed: 2023-06-29.

[2] Google deepmind. https://www.deepmind.com/. Accessed: 2023-06-29.

[3] John M. Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Zídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A A Kohl, Andy Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David A. Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. Nature, 596:583 – 589, 2021.

[4] Protein data bank (file format) - wikipedia. https://en.wikipedia.org/wiki/Protein_Data_Bank_(file_format). Accessed: 2023-06-27.

[5] Github - deepmind/alphafold: Open source code for alphafold. https://github.com/deepmind/alphafold. Accessed: 2023-06-20.

[6] Openfold: A faithful but trainable pytorch reproduction of deepmind's alphafold 2. https://github.com/aqlaboratory/openfold. Accessed: 2023-06-20.

[7] Docker: Accelerated, containerized application development. docker.com. Accessed: 2023-06-20.

[8] Ucsf chimera home page. https://www.cgl.ucsf.edu/chimera/. Accessed: 2023-06-21

[9] Openfold.ipynb - colaboratory. https://colab.research.google.com/github/aqlaboratory/openfold/blob/main/notebooks/OpenFold.ipynb. Accessed: 2023-06-21.

[10] Membrane protein. https://en.wikipedia.org/wiki/Membrane_protein. Accessed: 2023-06-17.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[12] https://www.alphafold.ebi.ac.uk/

Reasoning Over Description Logic-based Contexts with Transformers

Angelos Poulis

ABSTRACT

One way that the current state of the art measures the reasoning ability of transformer-based models is by evaluating accuracy in downstream tasks like logical question answering or proof generation over synthetic contexts expressed in natural language. However, all these contexts are in practice very simple; in most cases, they are generated from short first-order logic sentences with only a few logical operators and quantifiers. In this work, we construct a synthetic dataset over the description logic language $\mathcal{ALCQ}$ of 384K examples, which increases in two dimensions: i) reasoning depth, and ii) length of sentences. We show that the performance of our RoBERTa-based model, DELTA_M, is marginally affected when the reasoning depth is increased and it is not affected at all when the length of the sentences is increasing. We also paraphrase this dataset and we observe that when DELTA_M is fine-tuned, it performs equally well over the new dataset. Finally, in line with recent research findings, we observe that although transformer-based models seem to generalize well in increasing reasoning depths, this does not seem to be the case in decreasing reasoning depths.

Keywords: Natural Language Processing, Language Models, Description Logics, Knowledge Bases, Logical Reasoning.

ADVISORS

Manolis Koubarakis, Professor NKUA
Eleni Tsalapati, Postdoctoral Researcher

1 INTRODUCTION

Ontological knowledge is usually represented with formalisms based on description logic (DL) languages [1]. Description logics are fragments of first-order logic that can be used in knowledge-based systems to represent a domain of interest in a semantically well-defined manner. Their formal apparatus allows us to perform reasoning tasks such as consistency checking, deciding whether a sentence, or a set of sentences, logically implies another, and answering queries over knowledge bases encoded in DL languages. An expressive and decidable DL language that, besides the standard Boolean operators, supports also existential, universal, and numerical constraints is $\mathcal{ALCQ}$. For instance, one can formally express in $\mathcal{ALCQ}$ simple sentences like "*Anne is a female*" or "*Anne is a female and a postdoc*" or complex sentences such as "*If someone is a postdoc then she has a Ph.D. degree, she teaches at most two courses and she supervises at least one postgraduate student".* As a result of its expressive power, the complexity of reasoning in $\mathcal{ALCQ}$ is ExpTime-complete [19]. In this paper, we investigate the hypothesis that large language models (LLMs) [3] could be used to carry out reasoning tasks in the DL $\mathcal{ALCQ}$. If our hypothesis is correct then this would: i) enable the users to perform reasoning tasks (e.g., query answering) over ontological knowledge bases using natural language instead of logic language, and ii) possibly increase the time efficiency of the reasoning process. Recently, a substantial amount of research appeared in the literature investigating the ability of LLMs to carry out various reasoning tasks, such as logical question answering [2, 4], proof generation [13, 17] or satisfiability checking [12], over synthetic *contexts* expressed in natural language. These contexts are generated from propositional logic or selected fragments of first-order logic (FOL) expressed in natural language. As the focus is only on the reasoning ability of the models, the datasets are generated in such a way that world knowledge that LLMs have already learned is isolated. A standard way [2, 17] to achieve this is by performing random sampling over a probabilistic context-free grammar (PCFG). The correct answers are determined by a logical reasoner and the reasoning ability of the models is investigated with respect to the depth of the logical proof. In certain fragments of FOL, transformer-based models seem to have good performance, but in more expressive ones the models do not perform equally well [4, 18]. The

question that this work seeks to answer is *how well a transformer-based model will perform reasoning over an expressive DL language, like* $\mathcal{ALCQ}$. One question that arises here is if the fragment of the formal language is enough to measure the reasoning ability of a model. For instance, both synthetic sentences *"Anne admires Bob"* and *"If someone admires only people that are cold and not round or kind or that love at least one not green person, then they love someone that is cold or that loves someone kind"* can be expressed formally within $\mathcal{ALCQ}$. Hence, this study asks: If the context contains natural language sentences of the latter form, i.e., of high *linguistic complexity*, would the model perform with high accuracy? It is important to note that most of the benchmarks available in the literature are generated from short sentences whose formal representation contains only a few logical operators and quantifiers. We have constructed the dataset DELTA_D (DEscription Logics with TrAnsformers) of 384K question-answer examples (context-question-answer-depth) based on $\mathcal{ALCQ}$, where the question is the statement that we check whether it logically follows from the context and answer is "True" (if it does), "False" (if it conflicts with the context), "Unknown" (if none of the two). DELTA_D increases in complexity in both reasoning depth and breadth of sentences. An example of the dataset is presented in Figure 1.

Context:
If someone admires only people that are cold and not round or kind or that love at least one not green person, then they love someone that is cold or that loves someone kind.
If someone loves someone, then they love someone that is not big or not furry and that chases someone.
If someone loves someone that is not big or not furry and that chases someone, then they are red.
Anne admires Bob.
Dave loves someone cold and loves someone.
All individuals are different from each other.

Question:
Dave loves someone that is not big or not furry and that chases someone
Answer: **True**
Dave loves only people that are big and furry or that chase none
Answer: **False**
Anne is not red
Answer: **Unknown**

Fig. 5. An example from $\mathbf{DELTA}_D$ where the context contains long sentences, and the true and false sentences are of depth 2. The sentence with answer "unknown" does not logically follow from the context.

We test the performance of our transformer-based model, DELTA_M on two dimensions: i) the minimum reasoning depth required to obtain the answer, and ii) the complexity of the natural language sentences appearing in the context. We show that the performance of DELTA_M is marginally affected when the reasoning depth increases and it is not affected at all when the length of the sentences is increasing (accuracy 97.6% in max. reasoning depth and max. length of sentence). Hence, DELTA_M generalises well with respect to these two dimensions. We, also, tweaked a bit the probabilities of the PCFGs and we noticed that the accuracy of the model remained equally good. When DELTA_D is paraphrased, then the performance of DELTA_M drops (72.3% zero-shot accuracy). When it is fine-tuned on the paraphrased dataset then the accuracy of the model increases on both datasets (99.1% accuracy on the paraphrased data and 98.6% on the original data, for max depth and length). When we tested the fine-tuned model to a new dataset generated with the same process (i.e., from a PCFG grammar and then paraphrased), we observed that its generalisation ability improves (81.2% zero-shot accuracy). It was recently shown [15, 12, 23, 18, 14] that although transformer-based language models may perform well in reasoning tasks, they cannot acquire the logical principles governing the reasoning processes employed by standard logical reasoners. To further emphasize these results, and inspired by [18], we tested how a DELTA model performs on datasets with low reasoning depths when it is trained only on data of larger reasoning depths. In line with the literature, we observed that the accuracy of the model drops significantly to approximately 50%, which shows that, indeed, the model does not learn the underpinning reasoning rules but rather it *emulates* the logical QA task.

Overall, we make the following contributions:

1) We provide a large, balanced benchmark of 384K examples. Each example contains the context, a question, an answer (true, false, unknown), and the minimum reasoning depth required to obtain the answer. Additionally, the benchmark contains the respective data in formal form (expressed in $\mathcal{ALCQ}$) along with the minimum set of rules and facts involved in the proving process. This is a significant contribution because: (i) this is the first dataset of its kind for a DL, and (ii) building large benchmarks over expressive logic languages, like $\mathcal{ALCQ}$, is a challenging task as it requires

performing query answering with logic reasoners, a process that can be very time-consuming (~ 1 min. for KB with long rules/facts). Both the dataset and the code for its generation will be openly available to be used by other researchers.

2) We create the transformer-based model DELTA_M and we show that LLMs can perform reasoning over synthetic contexts generated from $\mathcal{ALCQ}$ sentences.
3) We show that the performance of LLMs is not affected by the length of the sentences.
4) We confirm recent research findings in that although LLMs may perform well in specific reasoning tasks, they have not learned the underpinning reasoning rules.
5) This is the first work in the literature that studies the ability of transformer-based models to perform reasoning tasks over expressive description logic languages.

2 BACKGROUND ON DESCRIPTION LOGICS

We can use $\mathcal{ALCQ}$ to represent knowledge about a domain by defining three types of entities: individuals (e.g., *John*), concepts (e.g., *Postdoc*, i.e., the concept describing the entities that are postdocs) and roles (e.g., *teaches*). *Concept expressions* can be formed using these entities, the Boolean constructors conjunction ($\sqcap$), disjunction ($\sqcup$), negation ($\neg$) and universal ($\forall$), existential ($\exists$), numerical ($\leq, \geq$) role restrictions. For instance, one can represent formally all entities that "have a Ph.D., teach at most two postgraduate courses and are not academics" ($\exists has.PhD \sqcap \leq 2\ teaches.PostgrCourse \sqcap \neg\ Academic$). *Rules* in $\mathcal{ALCQ}$ describe relationships between concept expressions. For example, the fact that all postdocs are described by the aforementioned concept expression can be represented with the concept inclusion $Postdoc \sqsubseteq \exists\ owns\ .PhD \sqcap \leq 2\ teaches.PostgrCourse \sqcap \neg\ Academic$. *Facts* describe knowledge about named individuals, i.e., that are *instances* of some concept (expression), e.g., John is a

Postdoc $(Postdoc(John))$, or how they related, e.g., John teaches Module05 $(teaches(John, Module05))$. An $\mathcal{ALCQ}$ *knowledge base* (KB) is a set of rules and a set of facts. The valid expressions in $\mathcal{ALCQ}$ are defined as follows. We start with a finite set of individuals, atomic concepts and roles. If A is an atomic concept and R is a role, according to the $\mathcal{ALCQ}$ grammar [1], the concept expressions C, D are constructed recursively as follows: $C, D ::= A \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall R.C \mid \exists R.C \mid \geq nR.C \mid \leq nR.C$, where the top concept $\top$ is a special concept with every individual as an instance, and the bottom concept $\bot$ is the dual of $\top$, i.e., with no individuals as instances. In $\mathcal{ALCQ}$ we can construct very complex concept expressions such as $\exists R_1.\Big(C \sqcup \big(\forall R_2.(D \sqcap \geq n\, R_3.\neg F)\big)\Big)$. A rule is of the form $C \sqsubseteq D$ and a *fact* of the form $C(a)$ or $R(a, b)$, where a, b individuals. Using complex expressions one can construct very complex rules and facts. We denote with *LHS* (left-hand side) the concept expression that appears on the left of the subsumption symbol ($\sqsubseteq$) in a rule and with *RHS* (right-hand side) the concept expression that appears on the right. The *inferred closure* of a KB $\mathcal{K}$ is the minimum set of rules and facts that can be logically inferred from $\mathcal{K}$. Given a KB $\mathcal{K}$ and a rule or a fact a, $depth(a, \mathcal{K})$ denotes the minimum number of rules and facts in $\mathcal{K}$ that can be used to logically deduce that a is true or false. Following the semantics of DLs, we make the *open-world assumption*, i.e., missing information is treated as unknown.

3 DATASET GENERATION

We investigate the ability of transformers to perform logical question answering over $\mathcal{ALCQ}$ KBs expressed in natural language with respect to three dimensions: i) the minimum depth of inference D a logic reasoner would need to answer the corresponding query, ii) the linguistic complexity level $\mathcal{L}$ of the knowledge required to answer the query, iii) paraphrased sentences. Each *example* in the dataset is a quadruple $\langle \mathcal{T}, \mathcal{Q}, \mathcal{A}, \mathcal{D} \rangle$, where $\mathcal{T}$ is the *context* containing $\mathcal{ALCQ}$ axioms expressed in natural language, $\mathcal{Q}$ a rule or fact expressed in natural language which forms the *question*, $\mathcal{A}$ is the *answer* which can be either *true*, *false*,

or *unknown*, and $\mathcal{D} = \text{depth}(q, K)$, where q is the query expressing formally the question $\mathcal{Q}$. The natural language of the dataset is English.

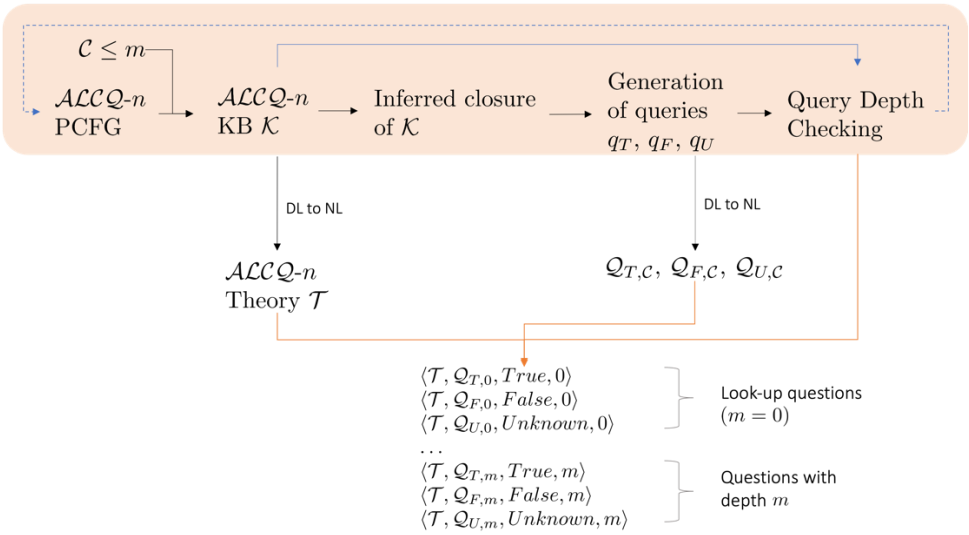


Fig. 6. Data generation pipeline for examples with n-level context and answers of minimum inference depth ≤ m

The pipeline for the generation of the dataset is presented in Figure 2. For the generation of an example of linguistic complexity level n ($L \leq n$) and number of inference steps up to m ($\mathcal{D} \leq \text{m}$), we first generate a small (with maximum 12 axioms and 14 facts) KB $\mathcal{K}$ using specially crafted PCFGs for n-level $\mathcal{ALCQ}$ KBs (denoted in Figure 2 with $\mathcal{ALCQ} - n$ PCFG). Then, the inferred closure of $\mathcal{K}$ is calculated by using and extending the reasoner HermiT [7], from which we calculate the queries. A KB $\mathcal{K}$ is kept only if it can produce queries with all three types of answers at all depths up to m, otherwise, it is discarded, and a new one is generated. Once this process is completed, the generated rules, facts along with the original KB $\mathcal{K}$, are translated into natural language statements and into the context $\mathcal{T}$, respectively, by utilizing a set of natural language templates.

3.1 KB Generation

We have defined two different pools of terms, *Pool A* and *Pool B*, from which we generate 40 datasets (20 from each pool) of 1000 KBs each, of different inference depths and axiom lengths. Pool A contains 14 atomic concepts, 5 roles, 8 individuals, all taken from RuleTaker dataset [2] (in RuleTaker the rules are simple

conjunctive implications, where the concept names are named “attributes”, the roles “relations” and the individual names “entities”). Pool B contains 8 atomic concepts, 8 roles and 8 individuals. For each pool of terms, we generate four types of $\mathcal{ALCQ}$-$\mathcal{L}$ KBs ($\mathcal{L} = 0,1,2,3$), based on the number of constructors and quantifiers appearing in their axioms. In general, an $\mathcal{L}$ KB includes axioms that their LHS or RHS contain $\mathcal{L}$ Boolean constructors and at most $\mathcal{L} + 1$ quantifiers, but also includes simpler axioms of smaller levels.

For instance, KBs of level $\mathcal{L} = 0$ contain only very simple facts or axioms that do not contain any Boolean constructors but can contain one quantifier, such as $Enthusiastic \sqsubseteq \exists\ supports\ .\ Enthusiastic$ (translated in NL as “Enthusiastic people support someone enthusiastic”), KBs of level $\mathcal{L} = 1$ contain axioms that their LHS or RHS can contain up to a single Boolean constructor and up to two quantifiers, etc. To keep the KBs processible by the reasoners, the axioms can contain up to seven atomic concepts and up to two nested quantifiers (e.g., $\exists likes.(\exists loves.(Cat))$, which describes the entities that like some entity that loves some cat). All KBs are checked with respect to satisfiability and consistency with HermiT. In line with [2, 17], each dataset contains questions whose answers have minimum depth of inference $\mathcal{D}$ ($\mathcal{D} = 0,1,2,3,5$).

3.2 Query Generation

For an inference depth D, a *true query* (answer=true) q is an axiom or fact selected from the inferred closure of a consistent $\mathcal{K}$, such that $depth(q, \mathcal{K}) = \mathcal{D}$. An *unknown query* (answer=unknown) is generated by creating a random fact or statement (using the corresponding PCFG) such that it does not belong to the inferred closure of $\mathcal{K}$ and is consistent with $\mathcal{K}$. A *false query* (answer=false) can be generated in three ways:

- From an inconsistent $\mathcal{K}$: for every $a \in \mathcal{K}$ if $\mathcal{K} \setminus \{a\}$ is consistent then a is a false query over the KB $\mathcal{K} \setminus \{a\}$.
- From a consistent $\mathcal{K}$: i) By negating a true query q with *depth*(q, $\mathcal{K}$) = D (and applying De Morgan's laws). ii) By automatically generating an appropriate axiom or fact a such that $\mathcal{K} \cup \{a\}$ is inconsistent and $depth(a, \mathcal{K}) = \mathcal{D}$. For instance, suppose that a KB $\mathcal{K}_1$ contains the axioms

$(\forall\, admires\,.\,\bot)(Anne)$ and $\forall\, admires\,.\,\bot \sqsubseteq \forall\, likes\,.\,Quiet$ which in natural language are translated into: "Anne admires none", "All people that admire none like only quiet people". Then, the fact $(\exists\, likes\,.\,\neg\, Quiet\,)(Anne)$ stating that "*Anne likes someone who is not quiet*" forms a false query for $\mathcal{K}$.

The disadvantage of the first approach is that it requires calling the reasoner multiple times, a time-consuming process, especially in KBs with long axioms (e.g., $\mathcal{L}$=3 KBs). Hence, we used the two latter approaches.

3.3 Data Translation to NL

The KBs and queries were translated to NL with the use of templates. The templates were created based on the user-friendly Manchester syntax for $\mathcal{ALCQ}$ [5]. Following this syntax, the intersection ($\sqcap$) and union ($\sqcup$) operators, are translated as "and" and "or", respectively, the existential ($\exists$) quantifier is translated as "someone" or "something" (depending on whether the pool is about people or things), the universal ($\forall$) as "only", and the numerical restrictions $\leq, \geq$ as "at most" and "at least". Also, we use the word "that" for intersections and nested quantifiers. For instance, the fact $\big(\exists\, likes\,.\,(\,\forall\, likes\,.\,Kind)\big)(\,Bob\,)$ is translated as "Bob likes someone that likes only kind people". Following the template-based approach suggested by [17], the axioms of the form $C \sqsubseteq D$ are, roughly, translated into NL in four different ways: i) "If C then D"; ii) "People/Things that are C are D"; iii) "All people/things that are C are D"; iv) If $C = \top$ and $D = \forall R.C'$ this is translated as "Someone/something can R only people/things that are C'". A fact $C(a)$ is translated as "a is C". To ensure that the resulting NL sentences are grammatically correct we have used a grammar checker (https://pypi.org/project/language-tool-python/).

3.4 The dataset $\mathbf{DELTA}_D$

After the generation and the translation of all KBs and queries, examples of the same depth and level from both pools are merged. This results in 20 datasets of 2000 KBs each, with each resulting dataset containing sentences from both vocabularies. From each KB we generated three queries (true, false, unknown) for each depth ($\mathcal{D} = \{0,1,2,3,5\}$), i.e., from each KB we generated $3 \times (d+1), d \in \mathcal{D}$.

So, in total, the dataset contains $\Sigma_{d\in\mathcal{D}} 3 \times (d+1) \times 2000 \times (\mathcal{L}_{max}+1)$, where $\mathcal{L}_{max} = 3$ is the highest of complexity levels.

3.5 Different Data Distributions

To check the performance of our model on different data distributions we generated new datasets by i) paraphrasing examples from DELTA_D, and ii) tweaking the PCFGs.

Paraphrasing examples. We initially used GPT-3 (text-davinci-003 model) and we tested the results by using the sentence transformer suggested by [11] with cosine similarity checking (t > 85%). However, the paraphrasing process was very time-consuming (> 10 seconds/KB) and the resulting data had only a few (~ 20%) acceptable alterations from the initial dataset. Hence, by observing the type of changes conducted on 730 axioms by GPT-3 we changed, also, the rest of the dataset. In particular, we used synonyms and opposites of terms appearing in the pools and we changed the tense of the verbs. All these changes were performed randomly, resulting in a dataset paraphrased by 83.14 %. So, for instance, a paraphrased version of the knowledge base $\mathcal{K}_1$ presented above is the following (the paraphrased terms are in italics).

- "Anne *looks up* to none."
- "If someone *does not admire anyone*, then they like only silent *individuals*."
- "If someone *is fond of* only *quiet* people, then they chase someone."

From which it is inferred that "Anne chases someone". Notice that although the KB is of the form $C(a)$, $C \sqsubseteq D$, $D \sqsubseteq F$, and the query is of the form $F(a)$, this is not so obvious in its paraphrased version unless the semantics of the words/phrases are taken into account.

Tweaking PCFGs. We increased the probability of $\forall$ from 0.33 to 0.70 and the probability of the disjunction from 0.50 to 0.80 at $\mathcal{L} = 3$ PCFG.

3.6 Statistical Features

As it is thoroughly discussed by [23], it is impossible to eliminate all statistical features that exist in data, besides, some of them inherently exist in logical reasoning problems (e.g., due to monotonicity, it is more likely that large KBs will produce positive examples). DELTA_D is balanced with respect to the following features: i) *KB size*: From the same KB we extract all three types of questions (true, false, unknown); ii) *Inference depth*: We keep a KB only if it can provide all three types of questions with the same inference depth; iii) *Formulation of the question*: The word "not" appears almost equal number of times in true questions (17.47 %), false questions (17.54 %) and unknown questions (15.54 %); iv) *Average length in words*: True questions 10.85, false questions 9.53, unknown questions 10.35.

4 EXPERIMENTS

For all our experiments we use RoBERTa-large, as the results from the literature [2, 18] showed that it has the best performance in QA over logical rules expressed in natural language. We train RoBERTa-large to predict true/false/unknown (i.e., multi-class classification) for each example. A context-question pair is supplied to the model as: [CLS] context [SEP] question [SEP]. For evaluation, we measure accuracy. The test data has an equal balance of True/False/Unknown answers, hence the baseline of random guessing is 33.3%. We use the AdamW optimizer [8] using its default values for betas and a weight decay setting of $1e^{-4}$. We take subsets of DELTA_D as they were formed during the generation process to train the model in the following way: The training is first performed per linguistic complexity level and then per depth (i.e., first it is trained in all linguistic complexity levels at depth 0, then in all complexity levels at depth 1, etc.). Hence, the final model DELTA_M has been trained to all depths and all linguistic complexity levels. The intermediate models are denoted with $\mathrm{DELTA}_{i,j}$ where i represents the reasoning depth and j the complexity level. For instance, the model $\mathrm{DELTA}_{3,2}$ has been trained to depths up to 3 and to all complexity levels on the previous depths and levels up to 2 on depth 3. $\mathrm{DELTA}_M = \mathrm{DELTA}_{5,3}$. For all

datasets, we partitioned the data into 70%/10%/20% splits for train/validation/test sets.

Table 1. Accuracy of DELTA models on Test (own), on $\mathbf{D_{5,3}}$ dataset, and on slices of $\mathbf{D_{5,3}}$ per depth.

	$\text{DELTA}_{0,3}$	$\text{DELTA}_{1,3}$	$\text{DELTA}_{2,3}$	$\text{DELTA}_{3,3}$	$\text{DELTA}_{5,3}$
Test (own)	99.7	99.3	99.0	99.2	97.4
$\text{D}_{5,3}$	68.4	87.2	83.7	98.7	98.4
$\mathcal{D}$ = N/A	95.6	98.6	98.4	99.0	98.9
$\mathcal{D} = 0$	99.9	100.0	100.0	100.0	99.1
$\mathcal{D} = 1$	57.5	99.4	99.4	99.2	98.2
$\mathcal{D} = 2$	61.9	81.7	97.6	97.9	97.2
$\mathcal{D} = 3$	43.8	71.9	54.8	98.8	98.0
$\mathcal{D} = 4$	39.3	70.1	54.4	98.6	98.7
$\mathcal{D} = 5$	27.0	66.1	52.0	97.4	97.6

Table 1 illustrates the performance of RoBERTa-large when trained on up to $\mathcal{L} \leq 3$ linguistic complexity over the various inference depths. For instance, the column $\text{DELTA}_{0,3}$ shows the performance of the model trained over all levels in depth 0. Test (own) represents the (held out) test set of the dataset that the model has been trained on. The $\text{D}_{5,3}$ dataset has questions from all inference depths ($\mathcal{D} \leq 5$) of all levels ($\mathcal{L} \leq 3$). "Depth N/A" refers to the unknown questions, as these are not provable. $\mathcal{D} = 0$ to $\mathcal{D} = 5$ represent the subsets of $\text{D}_{5,3}$ that correspond *only* to these depths, e.g., $\mathcal{D} = 3$ is the subset of $\text{D}_{5,3}$ containing only questions of depth 3. DELTA models have in general high accuracies but decrease as reasoning depths increase. They show improved but plateaued generalization on $\text{D}_{5,3}$, up to $\mathcal{D} \leq 5$. Interestingly, $\text{DELTA}_{1,3}$ generalizes better than $\text{DELTA}_{2,3}$. $\text{DELTA}_{3,3}$ not only matches the performance of $\text{DELTA}_{5,3}$ on $\text{D}_{5,3}$ but scores higher on most slices of $\text{D}_{5,3}$. This is an unexpected behavior of the model, as $\text{DELTA}_{5,3}$ has been trained at all depths and all linguistic complexity levels. For the first time, the Test (own) score drops below 99% with $\text{DELTA}_{5,3}$. This suggests that while $\text{DELTA}_{5,3}$ strives to excel at higher depths ($\mathcal{D} = 4$, $\mathcal{D} = 5$), it may compromise its generalization capability at lower depths.

Table 2. Accuracy of DELTA models on Test (own) across all levels.

	$\mathcal{D}=0$	$\mathcal{D}\leq 1$	$\mathcal{D}\leq 2$	$\mathcal{D}\leq 3$	$\mathcal{D}\leq 5$
$\mathcal{L}=0$	99.6	99.3	98.0	97.9	99.2
$\mathcal{L}\leq 1$	99.8	99.6	98.9	98.9	97.1
$\mathcal{L}\leq 2$	99.8	99.7	99.3	99.3	97.8
$\mathcal{L}\leq 3$	99.7	99.3	99.0	99.2	97.4

Table 2 demonstrates the performance of the intermediate models and the final model when tested on Test (own). For instance, the cell that corresponds to $\mathcal{D}=0$, $\mathcal{L}=0$ shows the accuracy of the model $\mathrm{DELTA}_{0,0}$. We observe that for depths $\mathcal{D}=0$ to $\mathcal{D}\leq 3$ the models are robust across levels, but when the depth increases to $\mathcal{D}\leq 5$ then increasing lengths affect performance (slightly).

Table 3. Accuracy of DELTA models on D ≤ 5 across all levels.

	$\mathcal{D}=0$	$\mathcal{D}\leq 1$	$\mathcal{D}\leq 2$	$\mathcal{D}\leq 3$	$\mathcal{D}\leq 5$
$\mathcal{L}=0$	64.5	68.8	83.8	96.1	93.6
$\mathcal{L}\leq 1$	70.7	82.5	87.7	97.9	98.0
$\mathcal{L}\leq 2$	61.7	86.9	87.0	98.6	98.4
$\mathcal{L}\leq 3$	68.4	87.2	83.7	98.7	98.4

Similarly, Table 3 illustrates the performance of the intermediate models $\mathrm{DELTA}_{i,j}$ on datasets of depth $\mathcal{D}\leq 5$ and level j. For instance, the cell that corresponds to $\mathcal{D}=0$, $\mathcal{L}=0$, shows the accuracy of model $\mathrm{DELTA}_{0,0}$ on data of level 0 and depth $\mathcal{D}\leq 5$. In practice, this table demonstrates, the progress of the model while training. We notice that its performance increases monotonically, except when it first sees data of next depths where the performance fluctuates, suggesting the different format of the data of larger lengths and smaller depths (e.g., $\mathcal{L}\leq 3$, $\mathcal{D}\leq 3$) compared to small lengths and larger depths (e.g., $\mathcal{L}=0$, $\mathcal{D}\leq 5$).

Zero-shot performance on paraphrased dataset. We paraphrased the dataset $\mathrm{D}_{5,3}$ based on the methodology described in Section 3.5. The zero-shot accuracy of DELTA_M over the paraphrased dataset was 72.3%, while the respective accuracy on the original $\mathrm{D}_{5,3}$ was 98.4%. This significant drop can be explained by the fact that the patterns of the original dataset have changed.

Zero-shot performance on slightly tweaked dataset. We changed the probabilities in the PCFG of linguistic complexity level 3 and we generated the new dataset DELTA_T (described in Section 3.5) of 1200 examples of up to reasoning depth $\mathcal{D} \leq 5$. This tweaking has resulted in sentences with 0.8/sentence disjunctions and 0.62/sentence universals. The accuracy of DELTA_M on the tweaked dataset was 94.5%, hence DELTA_M seems to be robust ($<$ 3% loss) over this new distribution.

Table 4. The performance of a model trained on questions of only depth 3, over datasets of various depths.

	D_0	D_1	D_2	D_3	D_4
DELTA_3	52.2	49.9	50.2	99.7	98.7

Zero-shot performance on smaller depths. We partitioned the dataset to the various depths, i.e., we extracted from DELTA_D datasets D_i, which contain only data of depth i (of all levels) and *not* up to i. We also trained the model DELTA_3 on 16K examples of only depth 3 in all lengths ($\mathcal{L} \leq 3$). The results are demonstrated in Table 4. We see that the model cannot generalize to decreasing depths but generalizes to increasing depths. This implies that the model has not learned the underpinning reasoning rules governing the reasoning process.

Table 5. Accuracy of $\mathbf{P-DELTA_{3,3}}$, $\mathbf{P-DELTA_{5,3}}$ models on paraphrased versions of Test (own), $\mathbf{D_{5,3}}$ and slices of $\mathbf{D_{5,3}}$ per depth.

	P-$\mathrm{DELTA}_{3,3}$	P-$\mathrm{DELTA}_{5,3}$
Test (own)	98.4	98.4
$\mathcal{D} \leq 5, \mathcal{L} \leq 3$	97.9	99.1
$\mathcal{D} = \mathrm{N/A}$	97.8	98.9
$\mathcal{D} = 0$	99.5	99.5
$\mathcal{D} = 1$	98.7	99.2
$\mathcal{D} = 2$	98.0	98.7
$\mathcal{D} = 3$	98.2	99.3
$\mathcal{D} = 4$	97.4	99.2
$\mathcal{D} = 5$	96.3	99.0

Performance of DELTA_M when fine-tuned to paraphrased data. We fine-tuned DELTA_M on the paraphrased versions of datasets of $\mathcal{D} \leq 3$ and $\mathcal{D} \leq 5$

(across all levels) of DELTA_D to construct the models $\mathrm{P}-\mathrm{DELTA}_{3,3}$, $\mathrm{P}-\mathrm{DELTA}_{5,3}$. Then, these models were tested over the paraphrased version of Test (own) and the paraphrased version of dataset $\mathrm{D}_{5,3}$. As it is shown in Table 5 (for $\mathcal{L} \leq 3$), fine-tuning the model on paraphrased datasets significantly improves its performance, particularly for $\mathcal{D} = 4, \mathcal{D} = 5$ questions, and boosts its zero-shot performance on the original $\mathrm{D}_{5,3}$ dataset.

Zero-shot performance on a new dataset. We created the dataset "Animals" from a new pool of terms that were about animals. As creating a new dataset from scratch with the methodology described in Section 3.5 required approximately 30 hours for 1000 KBs of $\mathcal{D} \leq 5$ and $\mathcal{L} \leq 3$, we replaced the terms in the original datasets $\mathcal{D} \leq 3$ and $\mathcal{D} \leq 5$ across all length levels with the corresponding terms from this pool of terms. After this, we paraphrased the resulting dataset. The accuracy of DELTA_M on the paraphrased dataset was 72.3% and the accuracy of $\mathrm{P}-\mathrm{DELTA}_{5,3}$ was 81.2%.

5 RELATED WORK

A survey of the most recent research on the use of transformers for reasoning tasks can be found in [22]. One of the first studies in the domain was from [2] with RuleTaker, where it was demonstrated the potential of transformers to perform logical question answering under CWA by training LLMs on synthetic datasets. However, their approach was limited to short expressions of simple conjunctive rules. [4], with the FOLIO dataset (1.4K), in the same line with RuleTaker, tested the ability of various LLMs for the same reasoning task under OWA. FOLIO was in FOL but without numerical restrictions. The result was that RoBERTa performed better than all models, but still the performance was low. [17] with the model ProofWriter built on top of T5 transformer [10] and the (modified) datasets from RuleTaker showed that LLMs can generate proofs (94.8% accuracy for depth 5). [9] showed that LLMs perform well (up to 90.5%) over context generated by propositional logic and a small subset of FOL. [18] introduced the much richer synthetic dataset LogicNLI (30K), under OWA for diagnosing LLMs behavior. They showed even the best performing model on

LogicNLI, RoBERTa, does not learn to reason and cannot generalize to different scenarios. [15] generated a very simple dataset (containing a single conjunction) for satisfiability checking and showed that models that perform well on hard problems do not perform equally well in easier problems, resulting in the conclusion that transformers cannot learn the underlying reasoning rules rather than they tend to overfit to patterns in the generated data. Also, [23, 18] achieved similar results. [14] showed that LLMs are good at producing valid individual steps but are struggling when there is more than one path in the proving process. Most of the aforementioned benchmarks are composed of short sentences. The ones that contain longer sentences (avg. 13 words/sentence) are small ($\leq$ 40K), while none of them have examples with numerical restrictions.

6 CONCLUSION AND FUTURE WORK

We generated the only large, balanced, synthetic dataset (384K) in the literature that targets expressive DLs (namely, $\mathcal{ALCQ}$), enjoys both high expressivity and high linguistic complexity and it is publicly available for further understanding of the functionality of LLMs. We showed that LLMs can carry out reasoning over expressive synthetic datasets with high accuracy. We, also, showed that the length of sentences in the context only marginally affects the performance of the model. Slight tweaking in the generation process did not affect the model, while it performed very well when fine-tuned to paraphrased datasets. In line with the recent results in the literature, we showed that our model has not learned the underpinning logic rules but can carry out reasoning tasks with high accuracy. As future work, we plan to explore the upper limit of the expressivity of the logic language that a transformer-based model will be able to perform reasoning tasks with high accuracy.

REFERENCES

[1] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

[2] Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2020. Transformers as soft reasoners over language. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 3882–3890. ijcai.org.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

[4] Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Luke Benson, Lucy Sun, Eka- terina Zubova, Yujie Qiao, Matthew Burtell, David Peng, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Shafiq Joty, Alexander R. Fabbri, Wojciech Kryscinski, Xi Victoria Lin, Caiming Xiong, and Dragomir Radev. 2022. Folio: Natural language reasoning with first-order logic.

[5] Matthew Horridge, Nick Drummond, John Goodwin, Alan L. Rector, Robert Stevens, and Hai Wang. 2006. The manchester OWL syntax. In *Proceedings of the OWLED*06 Workshop on OWL: Experiences and Directions, Athens, Georgia, USA, November 10-11, 2006, volume 216 of CEUR Workshop Proceedings. CEUR-WS.org.*

[6] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. 2008. Laconic and precise justifications in OWL. In *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings*, volume 5318 of *Lecture Notes in Computer Science*, pages 323–338. Springer.

[7] Ian Horrocks, Boris Motik, and Zhe Wang. 2012. The hermit OWL reasoner. In *Proceedings of the 1st International Workshop on OWL Reasoner Evaluation*

(ORE-2012), Manchester, UK, July 1st, 2012, volume 858 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[8] Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

[9] Santiago Ontañón, Joshua Ainslie, Vaclav Cvicek, and Zachary Fisher. 2022. Logicinference: A new dataset for teaching logical inference to seq2seq models. *CoRR*, abs/2203.15099.

[10] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

[11] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

[12] Kyle Richardson and Ashish Sabharwal. 2022. Pushing the limits of rule reasoning in transformers through natural language satisfiability. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 11209–11219. AAAI Press.

[13] Swarnadeep Saha, Sayan Ghosh, Shashank Srivastava, and Mohit Bansal. 2020. PRover: Proof generation for interpretable reasoning over rules. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 122–136, Online. Association for Computational Linguistics.

[14] Abulhair Saparov and He He. 2022. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *CoRR*, abs/2210.01240.

[15] Viktor Schlegel, Kamen V. Pavlov, and Ian PrattHartmann. 2022. Can transformers reason in fragments of natural language? In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing,

EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022, pages 11184–11199. Association for Computational Linguistics.

[16] Thomas Wolf Philipp Schmid Zachary Mueller Sourab Mangrulkar Sylvain Gugger, Lysandre Debut. 2022. Accelerate: Training and inference at scale made simple, efficient and adaptable. https: //github.com/huggingface/accelerate.

[17] Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. 2021. Proofwriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021*, volume ACL/IJCNLP 2021 of *Findings of ACL*, pages 3621–3634. Association for Computational Linguistics.

[18] Jidong Tian, Yitian Li, Wenqing Chen, Liqiang Xiao, Hao He, and Yaohui Jin. 2021. Diagnosing the first-order logical reasoning ability through logicnli. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 3738–3747. Association for Computational Linguistics.

[19] Stephan Tobies. 2000. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *J. Artif. Intell. Res.*, 12:199– 217.

[20] Jason Weston, Antoine Bordes, Sumit Chopra, and Tomás Mikolov. 2016. Towards ai-complete question answering: A set of prerequisite toy tasks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

[21] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pier- ric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System*

Demonstrations, pages 38–45, Online. Association for Computational Linguistics.

[22] Zonglin Yang, Xinya Du, Rui Mao, Jinjie Ni, and Erik Cambria. 2023. Logical reasoning over natural language as knowledge representation: A survey. *CoRR*, abs/2303.12023.

[23] Honghua Zhang, Liunian Harold Li, Tao Meng, Kai Wei Chang, and Guy Van den Broeck. 2022. On the paradox of learning to reason from data. *CoRR*, abs/2205.11502.