Sistem Tanya Jawab menggunakan Knowledge Graph mengenai Sistem Tata Surya

Jaish Muhammad[1], Kemas Rahmat Saleh Wiharja, S.T., M.Eng, P.hD[2]

[1,2]Fakultas Informatika, Universitas Telkom, Bandung
[1]jaishm@student.telkomuniversity.ac.id,
[2]bagindokemas@telkomuniversity.ac.id,

Abstrak

Manfaat dari *Knowledge Graph* (KG) bisa kita amati langsung, seperti optimalisasi search engine query Google, yang mempermudah dalam mencari sesuatu di internet. Diatas KG bisa juga dibangun sebuah *Question Answering System* (QA). Penelitian ini menggunakan artikel-artikel mengenai sistem tata surya dari halaman web NASA. Dengan menggunakan NLTK, artikel-artikel yang didapatkan dari halaman web NASA dipecah ke dalam bentuk *triple*. *Triple* tersebut kemudian diubah ke dalam bentuk *Knowledge Graph* yang disimpan di dalam Neo4j, kemudian dibangun *QA System* diatasnya. Proses validasi hasil sistem melibatkan ahli di bidang Astronomi. Hasil dari sistem ini adalah sistem yang menjawab query pertanyaan mengenai sistem tata surya. Performansi sistem diukur menggunakan akurasi, precision, recall, f1 score, dan mean reciprocal rank, yang mana didapatkan Akurasi = 0.78, Precision = 0.5, Recall = 1, F1 Score = 0.67, dan MRR = 0.2112955621.

Kata kunci : *Knowledge Graph*, *Question Answering system*, NASA, *triple*, sistem tata surya.

Abstract

The benefits of the Knowledge Graph (KG) can be observed directly, like the optimization of search engine queries like Google. which makes it easier to search for something on the internet. Based on the KG, a Question Answering System (QA) can also be built. This research uses articles about the solar system from NASA's website. Using NLTK, the articles obtained from NASA's website are broken down into triples. These triples are then transformed into a Knowledge Graph stored in Neo4j, and a QA System is built on top of it. The validation process of the system's results involves experts in the field of Astronomy. The outcome of this system is a system that answers queries about the solar system. The system's performance is measured using accuracy, precision, recall, F1 score, and mean reciprocal rank, which are obtained as follows: Accuracy = 0.78, Precision = 0.5, Recall = 1, F1 Score = 0.67, dan MRR = 0.2112955621.

Keywords: Knowledge Graph, Question Answering system, NASA, triples, solar system.

1. Pendahuluan

Latar Belakang

Sampai saat ini sudah cukup banyak penelitian yang menggunakan *Knowledge Graph* untuk *Question Answering System* [1][2][3][4][5][6][7][8][9][10]. Secara umum, *Knowledge Graph* menggambarkan suatu objek/entitas dan hubungan antara mereka [23]. *Knowledge graph* merupakan sebuah graf data yang dimaksudkan untuk mengumpulkan dan menyampaikan pengetahuan tentang dunia nyata, di mana nodenya mewakili suatu entitas dan edge mewakili hubungan yang berbeda antara entitas tersebut [24].

Fungsionalitas dari *Knowledge Graph* itu tidak terbatas, mulai dari answering search query di search engine, perbankan, retail, industri otomotif, industri perminyakan, kesehatan dan farmasi, penerbitan dan media [13]. Manfaat yang didapat dari *Knowledge Graph* antara lain lebih dari sekadar search, dapat memperoleh insight secara otomatis, rekomendasi secara otomatis, hingga analisis prediktif [14].

Implementasi dari *Question Answering system* tidak dibangun menggunakan *Knowledge Graph* saja, terdapat beberapa penelitian yang menggunakan Knowledge Base sebagai dasarnya. Sebuah KB-QA system mengambil natural language utterance sebagai input dan menghasilkan satu atau lebih jawaban sebagai output [15][16][17][18].

Berdasarkan studi literatur yang dilakukan, artikel-artikel yang didapatkan dari website NASA dapat diekstraksi dan dipecah ke dalam bentuk *triple*. *Triple* yang berhasil diekstraksi disimpan dalam bentuk *Knowledge Graph* menggunakan Neo4j. Kemudian, *Question Answering System* dibangun di atas *database* yang berhasil dibuat. Dataset yang digunakan berdasarkan halaman web Nasa yang memuat artikel mengenai sistem tata surya.

Topik dan Batasannya

Penelitian ini memiliki rumusan masalah sebagai berikut:

1. Bagaimana cara untuk membangun *Knowledge Graph* dari sebuah artikel?
2. Bagaimana cara untuk membangun *Question Answering system* menggunakan *Knowledge Graph*?

Batasan masalah:

1. Pengerjaan tugas akhir ini hanya sampai berhasil mengeluarkan jawaban dari pertanyaan yang diajukan.
2. Dataset dibatasi hanya dari satu sumber saja yaitu NASA, tidak terlalu luas.

Tujuan

Tujuan dari pelaksanaan tugas akhir ini adalah:

1. Mengetahui cara membangun sebuah *Knowledge Graph* dari sebuah artikel.
2. Mengetahui cara membangun *Question Answering system* menggunakan *Knowledge Graph*.
3. Berhasil mendapatkan jawaban yang relevan dari *Question Answering System* yang dibangun.

Organisasi Tulisan

Bagian lanjutan dari penelitian ini akan diisi dengan bagian 2 yaitu studi terkait, yang berisi studi-studi yang mendukung penelitian yang dilakukan. Selanjutnya pada bagian 3 merupakan sistem yang dibangun berdasarkan rancangannya. Pada bagian 4 dilakukan evaluasi dari sistem yang dibangun, terdapat penjelasan mengenai hasil pengujian sistem yang dibangun. Bagian 5 merupakan kesimpulan yang didapat dari hasil penelitian yang dilakukan, juga berisi saran yang bisa dilakukan untuk penelitian lanjutan.

2. Studi Terkait

2.1. Solar System

Solar System atau Tata Surya, merupakan kumpulan benda langit yang terdiri atas sebuah bintang yang disebut Matahari dan semua objek yang terikat oleh gaya gravitasinya [22]. Anggota utama dari tata surya terdiri dari Surya/Matahari, Merkurius, Venus, Bumi, Mars, Jupiter, Saturnus, Uranus, dan Neptunus[22]. Penelitian ini menggunakan artikel-artikel mengenai tata surya yang didapat dari laman web Nasa.

2.2. Knowledge Graph

Sebuah *knowledge graph* (i) sebagian besar menggambarkan entitas dunia nyata dan keterkaitannya, terorganisir dalam bentuk graf, (ii) mendefinisikan kelas dan relasi dari entitas yang memungkinkan dalam sebuah skema, (iii) memungkinkan untuk potensi *interrelating arbitrary entities* antar satu sama lain dan (iv) mencakup berbagai domain topik [12]. Pada penelitian ini, *Knowledge Graph* dibangun berdasarkan artikel-artikel dari halaman web Nasa, dan disimpan di dalam Neo4j.

2.3. NLTK

NLTK (Natural Language Toolkit), adalah sebuah rangkaian modul program sumber terbuka, tutorial, dan set masalah, yang menyediakan perangkat pengajaran linguistik komputasional yang siap digunakan [21]. Pada penelitian ini, NLTK digunakan sebagai library utama untuk proses ekstraksi *Triple* yang selanjutnya diubah ke dalam bentuk *Knowledge Graph*.

2.4. spaCy

spaCy adalah *library* Python gratis dan sumber terbuka yang menyediakan kemampuan canggih untuk melakukan pemrosesan bahasa alami (NLP) pada volume teks yang besar dengan kecepatan tinggi [25].

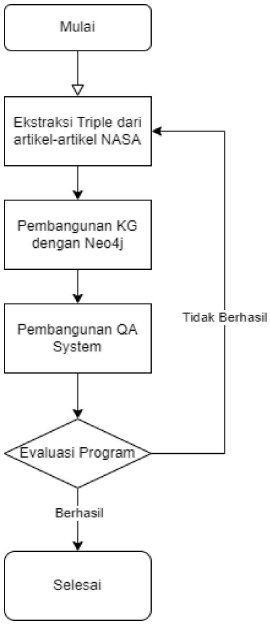
2.5. Named Entity Recognition (NER)

Named Entity Recognition (NER) merupakan metode pemrosesan bahasa alami (NLP) yang berfungsi untuk mendapatkan informasi dari sebuah kalimat. NER merujuk pada subjek utama dari sebuah teks, seperti nama, lokasi, perusahaan, acara dan produk, serta tema, topik, waktu, nilai moneter dan persentase [38].

2.6. Neo4j

Neo4j merupakan sebuah *database*, tetapi berbeda dengan *database* pada umumnya yang menyimpan data dalam baris, kolom dan tabel, Neo4j memiliki struktur yang fleksibel. Neo4j menyimpan relasi yang menghubungkan data-data. Dengan Neo4j, setiap data record atau node mengandung direct pointers ke seluruh node yang terkoneksi [19].

3. Sistem yang Dibangun



Gambar 1. Flowchart tahapan penelitian

Penjelasan tahapan-tahapan pada penelitian ini:

1. Proses Ekstraksi *Triple* dari artikel-artikel NASA

   Dari artikel-artikel NASA berikut, https://science.nasa.gov/sun/facts/, https://science.nasa.gov/solar-system/solar-system-facts/, https://science.nasa.gov/mercury/facts/, https://science.nasa.gov/venus/venus-facts/, https://science.nasa.gov/earth/facts/, https://science.nasa.gov/mars/facts/, https://science.nasa.gov/jupiter/jupiter-facts/, https://science.nasa.gov/saturn/facts/, https://science.nasa.gov/uranus/facts/, https://science.nasa.gov/neptune/facts/, https://science.nasa.gov/moon/facts/, dilakukan web scraping untuk kemudian dipecah ke dalam bentuk *Triple*. Pada penelitian ini penulis menggunakan NLTK sebagai alat NLP untuk mendeteksi *Triple* yang ada di dalam artikel-artikel tersebut, kemudian untuk subjek dan objek akan diperiksa menggunakan NER. Menggunakan bahasa pemrograman Python sebagai basisnya.

   Pada kode ekstraksi KG yang dibuat, terdapat fungsi *extract_data*, kode tersebut berfungsi untuk mendapatkan data dari artikel-artikel web yang dipakai, menggunakan *BeautifulSoup* untuk *scraping* data dari web, dan fungsi *sent_tokenize* dari nltk untuk memecah teks yang diekstraksi dari halaman web menjadi kalimat-kalimat individual. Hal tersebut dilakukan karena ekstraksi *triple* dilakukan pada tingkat kalimat.

2. Pembangunan *Knowledge Graph* dengan Neo4j

   Jika proses sebelumnya berhasil, akan diperoleh *triple* yang selanjutnya bisa dibangun sebuah *Knowledge Graph* dengan memasukkan data ke Neo4j. Selanjutnya dilakukan preprocessing dataset terlebih dahulu agar data akurat, dengan mengecek pola token kalimat dan menggunakan NER.

   Fungsi *extract_triple_with_context* bertujuan untuk mengekstraksi *triple* (subjek-predikat-objek) dari sebuah kalimat dengan mempertimbangkan pola *pos tag* dari token yang didapatkan menggunakan fungsi *pos_tag* dari NLTK, juga konteks predikat. *Triple* ini kemudian digunakan untuk membangun hubungan dalam *database* grafis Neo4j.

   Sebagai contoh ada kalimat “Earth orbits the Sun”, kalimat tersebut akan diproses menggunakan fungsi pos_tag NLTK, dan akan menghasilkan [('Earth', 'NN'), ('orbits', 'VBZ'), ('the', 'DT'), ('Sun', 'NNP')]. Sistem akan melakukan pengecekan untuk setiap tag dari hasil fungsi pos_tag. *Noun* pertama yang muncul akan diambil sebagai subjek, *verb* yang muncul setelah *noun* pertama akan diambil sebagai predikat, dan *noun* selanjutnya yang muncul setelah verb akan diambil sebagai predikat. *Triple* yang diekstrak akan menjadi: [("Earth", "orbits", "Sun", "Earth orbits the Sun")], dimana “Earth” merupakan subjek, “orbits” merupakan predikat, “Sun” merupakan objek, dan kalimat "Earth orbits the Sun" diambil sebagai konteksnya.

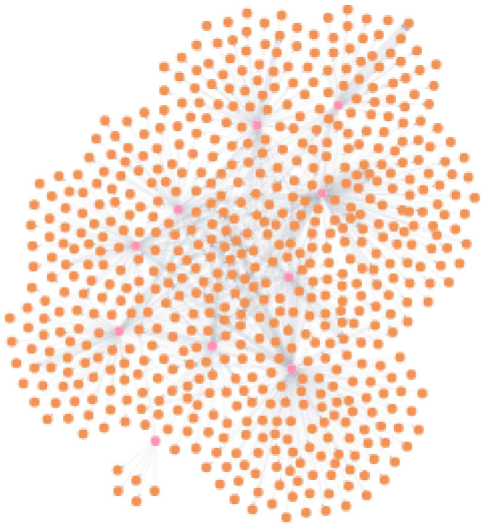
   Jika *triple* berhasil diekstraksi, subjek yang didapat akan dipilah, hanya akan melanjutkan proses jika subjek terdiri diantara 'Sun', 'Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune', dan 'Pluto'.

   Selanjutnya akan menjalankan *query cypher* yang akan menginput data *triple* yang sudah

didapatkan ke *database* neo4j.

Dataset *knowledge graph* yang berhasil dibuat memiliki 610 nodes, dengan 10 nodes subjek, dan 600 nodes objek. Pada dataset tersebut terdapat *triple* sebanyak 1,119 *triple*. Dengan menggunakan script Python, kita bisa mendapatkan jumlah fakta yang dipakai untuk mendapatkan *triple,* yaitu sebanyak 4,334 fakta.

Gambar 2. Tampilan *Knowledge Graph* yang terdapat pada neo4j dengan batas tampilan 25 *triples*.



Gambar 3. Tampilan KG secara keseluruhan.

3. Pembangunan *Question Answering System* berdasarkan KG

   Selanjutnya pembangunan *Question Answering system*, berbasis dari kode milik Bhavesh Bhatt [37]. Kode ini menggunakan Spacy dan NER untuk mendapatkan *triple* dari pertanyaan yang dikirim oleh user. Kemudian *triple* tersebut diolah menggunakan query cypher (cypher merupakan query dari neo4j) untuk mendapatkan data dari *database*. Jawaban didapatkan jika pencarian di *database* mengembalikan hasil.

   Dengan menggunakan *library* Spacy, sistem bisa mendapatkan *triple* yang berasal dari pertanyaan yang dikirim ke sistem.

   Fitur yang dibuat hanya dua jenis, pertama mencari objek dari *database* neo4j dengan pertanyaan yang mengandung subjek dan predikat. Kedua, mencari subjek dari *database* neo4j berdasarkan pertanyaan yang mengandung predikat dan objek.

4. Evaluasi *Question Answering system*

   Hasil dari *Question Answering system* dievaluasi dengan menghitung akurasi, *precision*, *recall*, *f1 score*, dan *mean reciprocal rank*.

4. Evaluasi

4.1 Kebenaran Sistem

Kebenaran sistem diuji dengan melibatkan ahli di bidang Astronomi. Pengujian dilakukan dengan membandingkan hasil jawaban dari query sistem dengan pendapat ahli. Hasil dari pengujian dapat dilihat di dalam tabel hasil uji berikut.

Tabel 1. Tabel hasil uji kebenaran sistem

No.	*Query*	*Output* Sistem	Jawaban Ahli	Penilaian Ketepatan Sistem
1	What is Sun?	The Sun is, 4.5 billion-year-old, in depth: Our Sun is a 4.5 billion-year-old yellow dwarf star – a hot glowing ball of hydrogen and helium – at the center of our solar system .	Basic answer should be: The Sun is a star, nearest to the Earth. Other answer is correct.	90%
2	What is Mercury?	The Mercury is, hydrogen, in depth Mercury 's exosphere is composed mostly of oxygen , sodium , hydrogen , helium , and potassium	Basic answer should be: Mercury is the smallest planet in the Solar System, nearest to the Sun. Other answer is correct.	70%
3	What is Venus?	The Venus is, solar system, in depth: Venus is one of only two planets in our solar system that does n't have a moon , but it does have a quasi-satellite that has officially been named Zoozve .	Basic answer: Venus is the second planet from the Sun, and our closest planetary neighbor. It's the hottest planet in our solar system, and is sometimes called Earth's twin. Other answer is correct.	70%
4	What is Earth?	The Earth is, center, in depth: Earth is composed of four main layers , starting with an inner core at the planet 's center , enveloped by the outer core , mantle , and crust .	Basic answer: Earth – our home planet – is the third planet from the Sun, and the fifth largest planet. It's the only place we know of inhabited by living things. Other answer is correct.	70%
5	What is Mars?	The Mars is, solar system, in depth: Mars is one of the most explored bodies in our solar system , and it 's the only planet where we 've sent rovers to roam the alien landscape .	Basic answer: Mars – the fourth planet from the Sun – is a dusty, cold, desert world with a very thin atmosphere. This dynamic planet has seasons, polar ice caps, extinct volcanoes, canyons and weather. Other answer is correct.	70%
6	What does Mars has?	The Mars has, center, in depth: Mars has a dense core at its center between 930 and 1,300 miles (1,500 to 2,100 kilometers) in radius .	The answer is correct	100%
7	What is Jupiter?	The Jupiter is, hydrogen, in depth: Like Jupiter , Saturn is mostly made of hydrogen and helium , the same two main components that make up the Sun .	Basic answer: It's the largest planet in our solar system – Jupiter is a gas giant. Jupiter's atmosphere is made up mostly of hydrogen (H2) and helium (He). The answer should be straight to the point (about Jupiter), not Saturn or other planets.	40%
8	What is Saturn?	The Saturn is, solar system, in depth: Saturn is the sixth planet from the Sun , and the second-largest planet in our solar system .	The answer is correct	100%
9	What is Uranus?	The Uranus is, hydrogen, in depth Uranus ' atmosphere is mostly hydrogen and helium , with a small amount of methane and traces of water and ammonia	Basic answer should be: Uranus is the seventh planet from the Sun, and it has the third largest diameter of planets in our solar system. Uranus appears to spin sideways. Other answer is ok	90%
10	What is Neptune?	The Neptune is, hydrogen, in depth Neptune 's atmosphere is made up mostly of hydrogen and helium with just a little bit of methane	Basic answer should be: Neptune is the eighth and most distant planet in our solar system. Other answer is ok	90%
11	What does Earth has?	The Earth has, moons, in depth: Earth 's Moon has no moons of its own .	The answer is correct, but trivial/no needs.	70%
12	What are Earth?	The Earth are, Earth, in depth: And while here on Earth the seasons are evenly spread over the year , lasting 3 months (or one quarter of a year) , on Mars the seasons vary in length because of Mars ' elliptical , egg-shaped orbit around the Sun	The answer is not related	40%
13	What planet orbits the sun?	The planet that orbits Sun are, Earth, Jupiter, Mercury, Venus, Mars, Saturn, Uranus, Neptune	Wrong sequence but name and quantity is correct.	50%
14	What does Earth orbits?	The Earth orbits, Sun, in depth: As Earth orbits the Sun , it completes one rotation every 23.9 hours .	The answer is correct	100%

15	What does Mars orbits?	The Mars orbits, Sun, in depth: As Mars orbits the Sun , it completes one rotation every 24.6 hours , which is very similar to one day on Earth (23.9 hours) .	The answer is correct	100%
16	What does Venus orbits?	The Venus orbits, Sun, in depth: Venus orbits the Sun from an average distance of 67 million miles (108 million kilometers), or 0.72 astronomical units.	The answer is correct	100%

Tabel 2. Data nilai hasil sistem

Penilaian Ketepatan Sistem	Label	Rank	Reciprocal Rank
90%	True Positive	6	0.1666666667
70%	False Positive	9	0.1111111111
70%	False Positive	10	0.1
70%	False Positive	11	0.09090909091
70%	False Positive	12	0.08333333333
100%	True Positive	1	1
40%	False Positive	15	0.06666666667
100%	True Positive	2	0.5
90%	False Positive	7	0.1428571429
90%	False Positive	8	0.125
70%	True Positive	13	0.07692307692
40%	False Positive	16	0.0625
50%	True Positive	14	0.07142857143
100%	True Positive	3	0.3333333333
100%	True Positive	4	0.25
100%	True Positive	5	0.2

1. Akurasi

Akurasi adalah proporsi dari prediksi yang benar terhadap total prediksi. Akurasi didapatkan dengan menjumlahkan kolom Penilaian Ketepatan Sistem dibagi dengan banyaknya nilai yang disertakan.

$$\text{Akurasi} = \frac{\text{Jumlah Prediksi Benar}}{\text{Total Prediksi}}$$

Berdasarkan hasil pengujian, didapatkan Akurasi = 78% atau **Akurasi = 0.78**.

2. Precision

Precision adalah proporsi dari prediksi yang benar terhadap total prediksi yang benar. Precision bisa dihitung dengan menjumlahkan True Positive dibagi dengan jumlah True Positive ditambah jumlah False Positive.

$$\text{Precision} = \frac{\text{Jumlah True Positive}}{\text{Jumlah True Positive} + \text{False Positive}}$$

Berdasarkan tabel Data hasil nilai sistem, bisa kita dapatkan jumlah True Positive = 8, dan jumlah False Positive = 8, maka Precision = 8/16 atau **Precision = 0.5.**

3. Recall

Recall adalah proporsi dari prediksi yang benar terhadap total kejadian sebenarnya yang benar. Recall bisa didapat dengan membagi jumlah True Positive dengan jumlah True Positive ditambah False Negative.

$$\text{Recall} = \frac{\text{Jumlah True Positive}}{\text{Jumlah True Positive} + \text{False Negative}}$$

Jumlah True Positive = 8, jumlah False Negative =0, Recall = 8/8 atau **Recall = 1**.

4. F1 Score

F1 Score adalah harmonic mean dari precision dan recall.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Precision = 0.5, Recall = 1, **F1 Score = 0.67**

5. Mean Reciprocal Rank (MRR)

Dari tabel nilai hasil sistem, bisa kita hitung MRR dengan menjumlahkan kolom *Reciprocal Rank* dan dibagi dengan banyaknya nilai.

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}.$$

Hasil yang didapatkan **MRR = 0.2112955621.**

4.2 Analisis Hasil Pengujian

Berdasarkan hasil pengujian kebenaran sistem, maka dapat dilakukan analisis sebagai berikut:

1. Berdasarkan dataset yang sudah didapat bisa kita hitung persentase jumlah triple dengan jumlah kalimat, dataset memiliki 610 *nodes*, dengan 10 *nodes* subjek, dan 600 *nodes* objek. Pada dataset tersebut terdapat triple sebanyak 1,119 triple. Dengan menggunakan script Python, kita bisa mendapatkan jumlah fakta yang dipakai untuk mendapatkan triple yaitu sebanyak 4,334 fakta. Persentase yang didapat sebesar 25.82%. Bisa disimpulkan bahwa masih banyak triple yang belum berhasil diekstraksi oleh sistem, karena masih ada 74.18% fakta yang terdapat pada halaman web Nasa yang belum bisa digunakan untuk penelitian ini.
2. Berdasarkan hasil pengujian sistem, bisa didapatkan Akurasi = 0.78, Precision = 0.5, Recall = 1, F1 Score = 0.67, dan MRR = 0.2112955621.

5. Kesimpulan

Hasil dari penelitian ini masih banyak yang perlu ditingkatkan, terutama di bagian ekstraksi *triple* yang masih banyak membutuhkan pengembangan, dimana *triple* yang didapat hanya berdasarkan *pos tag* dari NLTK dan *pattern recognition* sederhana, belum menggunakan learning untuk menyesuaikan konteks.

Fitur dari *Question Answering System* hanya mampu mendukung 2 jenis pertanyaan. Pertama pertanyaan deskriptif sederhana, seperti “What is Sun?” atau “What is Earth”, dan pertanyaan-pertanyaan sejenis, dimana pada pertanyaan tersebut akan dicari objek dari subjek “Sun” dengan predikat “is”. Kedua adalah pertanyaan yang mengandung objek dan predikat, seperti “What planet orbits the Sun?”, pada pertanyaan tersebut akan dicari subjek-subjek yang memiliki predikat “orbit” dan objek “Sun”.

Kekurangan dari penelitian ini diharapkan dapat ditingkatkan dan diselesaikan pada penelitian selanjutnya. Penelitian selanjutnya diharapkan bisa meningkatkan keakuratan fakta dari *Knowledge Graph* mengenai solar system dan juga menambah fitur pertanyaan yang bisa dijawab oleh *Question Answering System*.

Daftar Pustaka

[1] Philipp Christmann, Rishiraj Saha Roy, Abdalghani Abujabal, Jyotsna Singh, and Gerhard Weikum. 2019. Look before you Hop: Conversational Question Answering over Knowledge Graphs Using Judicious Context Expansion. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM '19). Association for Computing Machinery, New York, NY, USA, 729–738. DOI:https://doi.org/10.1145/3357384.3358016

[2] Abdalghani Abujabal, Rishiraj Saha Roy, Mohamed Yahya, and Gerhard Weikum. 2018. Never-ending learning for open-domain question answering over knowledge bases. In WWW.

[3] Hannah Bast and Elmar Haussmann. 2015. More accurate question answering on Freebase. In CIKM.

[4] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In EMNLP.

[5] Dennis Diefenbach, Pedro Henrique Migliatti, Omar Qawasmeh, Vincent Lully, Kamal Singh, and Pierre Maret. 2019. QAnswer: A Question Answering prototype bridging the gap between a considerable part of the LOD cloud and end-users. In WWW.

[6] Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. 2019. Knowledge graph embedding based question answering. In WSDM.

[7] Thomas Pellissier Tanon, Marcos Dias de Assuncao, Eddy Caron, and Fabian M Suchanek. 2018. Demoing Platypus – A multilingual question answering platform for Wikidata. In ESWC.

[8] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. 2012. Template-based question answering over RDF data. In WWW.
[9] Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, and Gerhard Weikum. 2013. Robust question answering over the web of linked data. In CIKM.
[10] Abdalghani Abujabal, Rishiraj Saha Roy, Mohamed Yahya, and Gerhard Weikum. 2017. QUINT: Interpretable Question Answering over Knowledge Bases. In EMNLP.
[11] Chowdhury, Sudip. (2019). Knowledge Graph: The Perfect Complement to Machine Learning. [Online] available at: https://towardsdatascience.com/knowledge-graph-bb78055a7884
[12] Paulheim, H. (2017). Knowledge graph refinement: A survey of approaches and evaluation methods. Semantic Web, 8(3):489–508.
[13] Nigam, V. V., Paul, S., Agrawal, A. P., and Bansal, R. (2020). A review paper on the application of knowledge graph on various service providing platforms. In 2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence), pages 716–720. IEEE.
[14] Simone, S. (2020). Using knowledge graphs and search for decision intelligence. KMWorld.
[15] Hannah Bast and Elmar Haussmann. 2015. More accurate question answering on freebase. In CIKM.
[16] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In EMNLP.
[17] Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. TACL.
[18] Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In ACL.
[19] Swalin, Alvira. (2018). Building a Question-Answering System from Scratch— Part 1. [Online] available at: https://towardsdatascience.com/building-a-question-answering-system-part-1-9388aadff507
[20] Gao, J., Li, X., Xu, Y.E., Sisman, B., Dong, X.L., Yang, J.: Efficient knowledge graph accuracy evaluation. Technical Report (2019). https://users.cs.duke.edu/~jygao/KG_eval_vldb_full.pdf
[21] Loper, E., & Bird, S. (2002). Nltk: The natural language toolkit. arXiv preprint cs/0205028.
[22] Harris, M. (2023, June 27). Sistem Tata Surya: Definisi, Teori, dan Sistem Penyusunnya. Gramedia Literasi. https://www.gramedia.com/literasi/sistem-tata-surya/
[23] Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. 2019. Industry-scale knowledge graphs: lessons and challenges. Commun. ACM 62, 8 (August 2019), 36–43. https://doi.org/10.1145/3331166
[24] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. 2021. Knowledge Graphs. ACM Comput. Surv. 54, 4, Article 71 (May 2022), 37 pages. https://doi.org/10.1145/3447772
[25] What is spaCy? | Domino Data Lab. (n.d.). https://domino.ai/data-science-dictionary/spacy
[26] Sun: Facts - NASA Science. (n.d.). https://science.nasa.gov/sun/facts/
[27] Mercury: Facts - NASA Science. (n.d.). https://science.nasa.gov/mercury/facts/
[28] Venus: Facts - NASA Science. (n.d.). https://science.nasa.gov/venus/venus-facts/
[29] Facts About Earth - NASA Science. (n.d.). https://science.nasa.gov/earth/facts
[30] Mars: Facts - NASA Science. (n.d.). https://science.nasa.gov/mars/facts/
[31] Jupiter: Facts - NASA Science. (n.d.). https://science.nasa.gov/jupiter/jupiter-facts/
[32] Saturn: Facts - NASA Science. (n.d.). https://science.nasa.gov/saturn/facts/
[33] Uranus: Facts - NASA Science. (n.d.). https://science.nasa.gov/uranus/facts/
[34] Neptune: Facts - NASA Science. (n.d.). https://science.nasa.gov/neptune/facts/
[35] Moon Facts - NASA Science. (n.d.). https://science.nasa.gov/moon/facts/
[36] Solar System: Facts - NASA Science. (n.d.). https://science.nasa.gov/solar-system/solar-system-facts/
[37] Bhattbhavesh. (n.d.-b). neo4j-tutorials/2-movie-q-and-a-neo4j-notebook.ipynb at main · bhattbhavesh91/neo4j-tutorials. GitHub. https://github.com/bhattbhavesh91/neo4j-tutorials/blob/main/2-movie-q-and-a-neo4j-notebook.ipynb
[38] Barney, N. (2023, March 10). named entity recognition (NER). WhatIs. https://www.techtarget.com/whatis/definition/named-entity-recognition-NER

Lampiran

1. Kode Python untuk ekstraksi triple dan membuat Knowledge Graph dengan mengirimkan hasil triple ke Neo4j.

```
#Knowledge Graph Extractor
import nltk
import spacy
import requests
from bs4 import BeautifulSoup
from neo4j import GraphDatabase
from nltk.tokenize import sent_tokenize, word_tokenize

nlp = spacy.load("en_core_web_sm")

# Download NLTK resources
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

# Neo4j connection details
uri = ""
username = ""
password = ""

# Connect to Neo4j
driver = GraphDatabase.driver(uri, auth=(username, password))

# Function to execute Cypher query
def run_cypher_query(query, parameters=None):
    with driver.session() as session:
        result = session.run(query, parameters)
        return result.data()

# Function to parse and extract information from a given URL
def extract_data(url):
    with driver.session() as session:
        # Get page content
        response = requests.get(url)
        page_content = response.content

        # Parse HTML
        soup = BeautifulSoup(page_content, 'html.parser')

        # Extract relevant information from the page
        text = ' '.join([paragraph.text.strip() for paragraph in soup.find_all('p')])

        # Tokenize text into sentences
        sentences = sent_tokenize(text)

        # Extract triples with context and create relationships
        for sentence in sentences:
            triples = extract_triples_with_context(sentence)
            for triple in triples:
                subject, predicate, obj, predicate_context = triple
                if subject in ['Sun','Mercury','Venus','Earth','Mars','Jupiter','Saturn','Uranus','Neptune','Pluto']:
                  print(triple)
                  session.run("""
                      MERGE (subject:Subject {name: $subject})
                      MERGE (object:Object {name: $object})
                                          MERGE (subject)-[:RELATION {predicate: $predicate, predicate_context:
$predicate_context}]->(object)
```

```
            """, subject=subject, object=obj, predicate=predicate, predicate_context=predicate_context)

# Function to extract subject-predicate-object triples from a sentence with context
def extract_triples_with_context(sentence):
    triples = []

    # Tokenize sentence into words
    words = word_tokenize(sentence)

    # Part-of-speech tagging
    pos_tags = nltk.pos_tag(words)

    # NER with spaCy
    doc = nlp(sentence)
    entities = {ent.text: ent.label_ for ent in doc.ents}

    used_words = set()

    # Find indices of nouns, verbs, adjectives, and adverbs
    noun_indices = [i for i, (word, pos) in enumerate(pos_tags) if pos.startswith('NN')]
    verb_indices = [i for i, (word, pos) in enumerate(pos_tags) if pos.startswith('VB')]
    adj_indices = [i for i, (word, pos) in enumerate(pos_tags) if pos.startswith('JJ')]
    adv_indices = [i for i, (word, pos) in enumerate(pos_tags) if pos.startswith('RB')]

    # Extract triples using nouns as subjects, verbs as predicates, and nouns, adjectives, or adverbs as objects
    for noun_index in noun_indices:
        subject = words[noun_index]
        # Refine subject if available in NER
        if subject in entities:
                subject = next((ent for ent, label in entities.items() if label in ['PERSON', 'ORG', 'GPE', 'LOC',
'PRODUCT']), subject)

        for verb_index in verb_indices:
            if words[verb_index] != '–':
                predicate = words[verb_index]
                for i in range(verb_index + 1, len(words)):
                    if i > noun_index:
                        # Check if the word is a noun, adjective, adverb, or a cardinal number followed by a qualifier
                        if (pos_tags[i][1].startswith('NN') or
                            pos_tags[i][1].startswith('JJ') or
                            pos_tags[i][1].startswith('RB') or
                            (pos_tags[i][1] == 'CD' and i < len(pos_tags) - 1 and
                            (pos_tags[i + 1][1].startswith('JJ') or
                              pos_tags[i + 1][1].startswith('NN') or
                              pos_tags[i + 1][1].startswith('RB')))):
                            if words[i] != '–':
                                if words[i] not in used_words:
                                    obj = words[i]
                                    used_words.add(words[i])
                                    j = i + 1
                                    if pos_tags[i][1] == 'CD' and i < len(pos_tags) - 1:
                                        obj += ' ' + words[i + 1]  # Include the next word if it's part of the object
                                        used_words.add(words[i + 1])
                                        j += 1
                                    while j < len(pos_tags) and pos_tags[j][1].startswith('NN'):
                                        if words[j] != '–':
                                            obj += ' ' + words[j]
                                            used_words.add(words[j])
                                        j += 1
                                    # Refine object if available in NER
                                    if obj in entities:
                                            obj = next((ent for ent, label in entities.items() if label in ['PERSON', 'ORG', 'GPE',
```

```
'LOC', 'PRODUCT']), obj)
                        # Extract context words around predicate
                        predicate_context = ' '.join(words)
                        triples.append((subject, predicate, obj, predicate_context))
    return triples

# List of URLs to scrape
urls = [
    "https://science.nasa.gov/sun/facts/",
    "https://science.nasa.gov/solar-system/solar-system-facts/",
    "https://science.nasa.gov/mercury/facts/",
    "https://science.nasa.gov/venus/venus-facts/",
    "https://science.nasa.gov/earth/facts/",
    "https://science.nasa.gov/mars/facts/",
    "https://science.nasa.gov/jupiter/jupiter-facts/",
    "https://science.nasa.gov/saturn/facts/",
    "https://science.nasa.gov/uranus/facts/",
    "https://science.nasa.gov/neptune/facts/",
    "https://science.nasa.gov/moon/facts/"
]

# Extract data from each URL and import it into Neo4j
for url in urls:
    extract_data(url)

# Close Neo4j driver
driver.close()
```

2. Kode untuk *Question Answering System*.

```
#Question Answering System
from neo4j import GraphDatabase
import spacy
import gradio as gr

nlp = spacy.load("en_core_web_sm")

# Neo4j connection details
uri = ""
username = ""
password = ""

# Connect to Neo4j
driver = GraphDatabase.driver(uri, auth=(username, password))

def extract_entity(question):
    doc = nlp(question)
    subject = None
    relation = None
    obj = None

    # Using named entity recognition to identify potential subjects and objects
    entities = {ent.text: ent.label_ for ent in doc.ents}

    # Using dependency parsing to find the main verb and its direct objects and subjects
    for token in doc:
        if token.dep_ in ['ROOT']:
            relation = token.head.text
            for child in token.children:
                if child.dep_ in ['nsubj']:
                    subject = child.text
```

```
            elif child.dep_ in ['dobj', 'attr', 'prep']:
                obj = child.text

    # If named entities are found, refine the subject and object from the entities
    if subject in entities:
        subject = next((ent for ent, label in entities.items() if label in ['PERSON', 'ORG', 'GPE', 'LOC', 'PRODUCT']),
subject)
    if obj in entities:
        obj = next((ent for ent, label in entities.items() if label in ['PERSON', 'ORG', 'GPE', 'LOC', 'PRODUCT']), obj)

    return subject, relation, obj

def chatbot(input, history=[]):
    output = get_answer(input)
    history.append((input, output))
    return history, history

def get_answer(question):
    doc = nlp(question)
    subject, relation, obj = extract_entity(question)
    print(f"Subject: {subject}, Relation: {relation}, Object: {obj}")

    with driver.session() as session:
        # Handle case where the subject, relation, and object are all present
        if relation and (obj in ['What','When','Why','Where','Who','How']):
            query = f"MATCH (subject:Subject {{name: '{subject}'}})-[r:RELATION {{predicate:
'{relation}'}}]->(object:Object) RETURN subject.name AS subject_name, object.name AS object_name,
r.predicate_context"
            result = session.run(query)
            entities = [f"The {subject} {relation}, {record['object_name']}, in depth: {record['r.predicate_context']}" for
record in result]
            if entities:
              return entities
            else:
              return f"No information found for the {subject}."

        # Handle case where only relation and object are present (e.g., "What are the planets that orbit the Sun?")
        elif relation and obj:
            print(relation, obj)
            query = f"""
                MATCH (s:Subject)-[r:RELATION {{predicate: '{relation}'}}]->(o:Object {{name: '{obj}'}})
                RETURN s.name AS subject_name, o.name AS object_name
            """
            result = session.run(query)
            entities = [f", {record['subject_name']}" for record in result]
            if entities:
                return f"The {subject} that {relation} {obj} are" + "".join(entities)
            else:
                return f"No information found for entities that {relation} {obj}."
        return "Sorry, I don't understand what you're asking."

gr.Interface(fn=chatbot,
             inputs=["text", 'state'],
             outputs=["chatbot", 'state']).launch(debug=True)
```

3. Ahli yang dilibatkan