Leveraging OpenAPI for Microservice Decomposition: A Comparative Study on Features, Encodings and Algorithms on a real MES

Michael Oberle
Competence Centre for Digital Tools in Manufacturing
Fraunhofer Institute for Manufacturing Engineering
Stuttgart, Germany
michael.oberle@ipa.fraunhofer.de

Thomas Bauernhansl
Institute of Industrial Manufacturing and Management
University of Stuttgart
Stuttgart, Germany
thomas.bauernhansl@iff.uni-stuttgart.de

***Abstract*—The demand for greater modularity and scalability within software architecture drives the transition from monolithic systems to microservices. This paper delves into decomposing a real-world monolithic Manufacturing Execution System (MES) into microservices, focusing on semantic analysis using word encodings applied to 326 OpenAPI endpoints. We studied the results of a machine learning approach in comparison to results obtained from human experts. We evaluated the impact of encoding types, features and algorithms on segmentation results. We compared different feature combinations, and classic encodings such as TF-IDF, word2vec and fastText embeddings with openAI embeddings and custom-trained embeddings, and also non-centroid algorithms with k-means. In our real-world scenario, we have found that the best combinations produce good results with ARI and NMI scores at 0.75 and above compared to human experts' ground truth. However, the low silhouette scores below 0.3 in the same runs indicate the limitations of the method. The method facilitates the decomposition process but requires human-driven configuration and verification of results.**

***Index Terms*—Microservices, Decomposition, OpenAPI, Semantic Analysis, Manufacturing Execution System**

I. Introduction

Microservice architectures have become a widely popular architecture style for many applications. At its core, a microservice architecture decomposes a software application into small, independent services that communicate with each other through lightweight mechanisms such as HTTP RESTful APIs. Each service is responsible for a distinct functionality and can be developed, deployed, and scaled independently. This stands in contrast to the monolithic architectural style where an application is built as a single, indivisible unit [1]. The surge in popularity can be attributed to several advantages over traditional monolithic designs. Microservice architectures promote modularity, scalability of individual components, improved maintainability, and faster development cycles. They also enable teams to work on different services simultaneously, which boosts productivity. Furthermore, microservice architectures enable resilience. The failure of one service does not necessarily result in the collapse of the whole system [2].

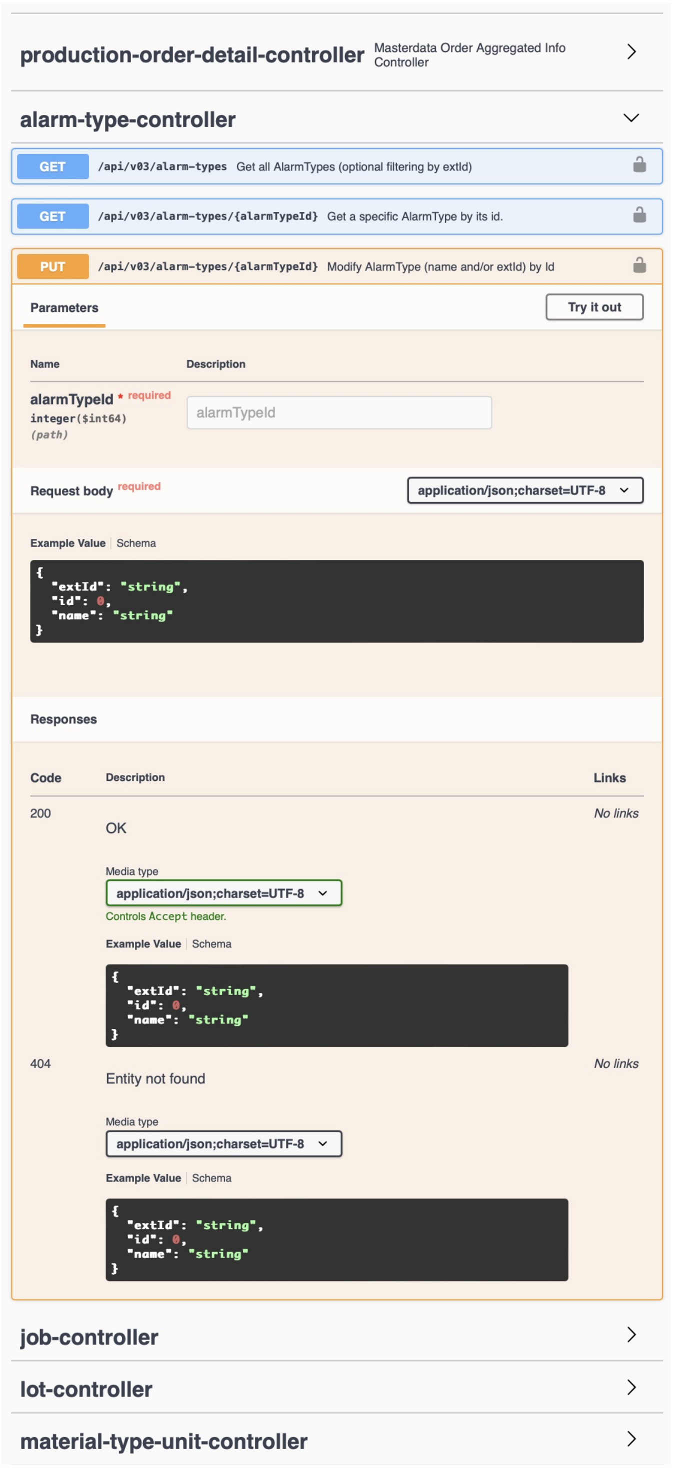
This work was supported by a grant from the German Federal Ministry of Education and Research for the project "DigiBattPro 4.0" under grant 03XP0374C.

Therefore, much attention has been given to breaking down monolithic applications into a microservice architecture in the software engineering research community [3]. A particular area of interest has been semantic methods for decomposing monoliths based on artefacts such as openAPI specifications or software engineering models [4]. The results of the techniques are promising. However, the research typically focuses on standard examples for consistency and comparability. The effectiveness of semantic approaches in the complexity of real scenarios has rarely been investigated. The paper aims to close this gap by examining the performance of a semantic OpenAPI-based decomposition technique in a real-world scenario, specifically a Manufacturing Execution System (MES).

The paper is structured as follows: Section 2 explores related work covering microservice architectures in MES and the existing work on semantic microservice decomposition techniques. Section 3 introduces the real-world case study, and section 4 describes our experiment design. Our findings are detailed in Section 5. Section 6 offers a comprehensive discussion of the results and their implications. The paper concludes with Section 7, where we summarize primary findings and suggest future research directions.

II. Related Work

Techniques for microservice decomposition can be distinguished by the type of input the technique is based on. Static analysis uses the application source code to propose microservice candidates. For example, Kamimura et al. [5] created a method for extracting microservice candidates from source code using a clustering algorithm. Dynamic analysis uses the application's behavior in a real-world performance for the decomposition analysis. An example is an approach analyzing the access logs of monolithic applications through unsupervised machine learning [6]. Using software engineering artefacts instead of the source code is a special variant of static analysis. It leverages API descriptions, data flow diagrams, class diagrams, or other models for the analysis


production-order-detail-controller Masterdata Order Aggregated Info Controller

alarm-type-controller

GET /api/v03/alarm-types Get all AlarmTypes (optional filtering by extId)

GET /api/v03/alarm-types/{alarmTypeId} Get a specific AlarmType by its id.

PUT /api/v03/alarm-types/{alarmTypeId} Modify AlarmType (name and/or extId) by Id

Parameters — Try it out

Name	Description
alarmTypeId * required integer($int64) (path)	alarmTypeId

Request body required — application/json;charset=UTF-8

Example Value | Schema

```
{
  "extId": "string",
  "id": 0,
  "name": "string"
}
```

Responses

Code	Description	Links
200	OK	No links
404	Entity not found	No links

200 — Media type: application/json;charset=UTF-8

Controls Accept header.

Example Value | Schema

```
{
  "extId": "string",
  "id": 0,
  "name": "string"
}
```

404 — Media type: application/json;charset=UTF-8

Example Value | Schema

```
{
  "extId": "string",
  "id": 0,
  "name": "string"
}
```

job-controller

lot-controller

material-type-unit-controller

Fig. 1. Excerpt from the Xetics REST API

to suggest microservice candidates. This has the advantage that they can be used before the application is coded. Gysel et al. created Service Cutter, a framework for microservice decomposition using domain models and use cases to extract coupling information, represented as weighted graphs to allocate closely related services [7]. They then utilized clustering algorithms to identify microservice candidates. Baresi et al. proposed a technique that uses the semantic similarity between Open API operations [8]. They used schema.org as a vocabulary reference to map OpenAPI specifications and calculate the distributional similarity between the operation words. As a result, OpenAPI specifications that were semantically similar were grouped together to create candidate microservices. Al-Debagy et al. also leverage the OpenAPI specifications for finding similar words in the operation names [9]. They use word embedding models such as fastText to find semantic similarities between operation names and group semantically similar operation names in a microservice candidate through hierarchical clustering. They extended their work evaluating the microservice candidates with evaluation cohesion and complexity metrics to further fine-tune the microservice decomposition [10]. They evaluated their algorithm through two standard examples of microservice applications (Kanban Dashboard and Money Transfer), which also serve as a test for [8]. However, an investigation of its effectiveness on a real-world system and the utility of the approach is yet missing. Furthermore, it is not yet clear what combination of feature, encoding, and embedding will result in successful outcomes as only Affinity Propagation algorithm, a single feature and one fastText and word2vec embedding are considered.

To address this gap, our study was guided by the following research questions:

Q1. *How well does the approach perform in a real-world scenario compared to human software engineering experts?*
Q2. *Which attributes in the OpenAPI specification yield the most effective results?*
Q3. *Does the way we encode features affect the results?*
Q4. *Does the incorporation of a domain-specific word embedding enhance the results?*
Q5. *How do clustering algorithms differ in terms of microservice candidates' quality?*

III. Real-world Case: Decomposition of a Manufacturing Execution System

Our study aims to investigate our research questions by decomposing a monolithic real-world MES into microservice candidates. A MES is software that connects plant operations with enterprise systems, allowing real-time visibility into production processes. It serves as a centralized platform that manages and monitors work-in-progress on a factory floor, facilitating optimized production, enhanced scheduling, efficient resource allocation, and superior product quality. The MES offers a wide range of functionalities, including product lifecycle management, process control, data analysis, quality control, and more. The MES building the case for evaluating the microservice decomposition technique is the Xetics Lean MES. It is a state-of-the-art MES emphasizing production management flexibility and efficiency. It is designed primarily for assembly shops but can also be used in other production areas. The system offers a number of functionalities to support shop floor operations, which are listed and described in Table

I. The system is built as a monolith. However, it offers an extensive REST API that includes CRUD and other state change operations for all significant entities related to the system's functionalities. Some of these entities include orders, production lots, production plans, shifts, media, and material stock. The REST API is described according to the OpenAPI specification including fully specified endpoints, with operation description, input parameters, and response. A part of the API as displayed in Swagger can be seen Figure 1. The API includes over 326 endpoints, which are grouped by 58 tags (e.g., "alarm-type-controller").

TABLE I
DESCRIPTION XETICS LEAN MES FUNCTIONALITIES

Functionality	**Description**
production process plan modeling	Modelling of the sequence of steps and process parameters required for a product to be completed.
production order management	Creation, release of orders for a specific product, and monitoring of the production process.
production execution	Tracking of each resource material and process parameter used for creating a product along the process plan.
skill management	Management of available skill resources and enforcing of skill requirements required by the process.
shift management	Rostering of labor shifts for the production plant.
production planning	Scheduling of production orders under consideration of maximum resource utilization and order due dates.
equipment integration	Collection of data from equipment at different manufacturing stations.
material management	Stock management of consumable material required in the production process.
document management	Storage and versioning of documents required in the production process such as Standard Operation Procedures or other manuals.

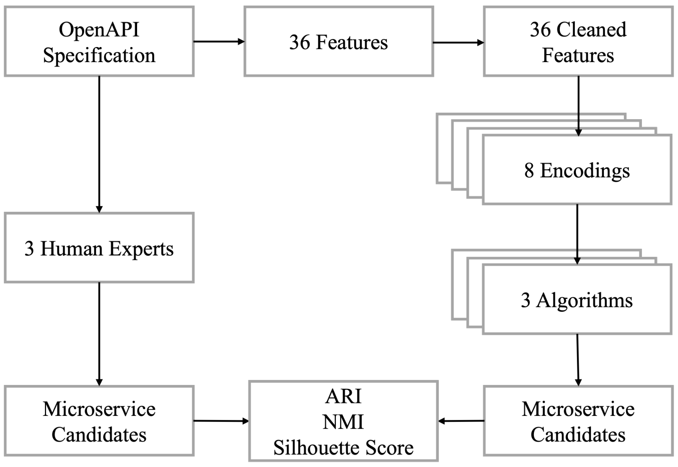


Fig. 2. Experiment Design Overview

TABLE II
DESCRIPTION OF FEATURES IN THE OPENAPI SPECIFICATION

Feature	**Description**
operation id	The id uniquely identifying the endpoint.
tag	Descriptor categorizing the endpoint. In case of Xetics MES this is the controller class name for the endpoint.
path	Path for accessing the endpoint.
method	Http method type for the endpoint (e.g. PUT, GET, DELETE...).
summary	Descriptive summary of the purpose and consequences of the API.
parameter 1-11	Name of parameters in query.
parameter 1-11 description	Description of parameters in query.
request body	Object in request body of the endpoint.
response 1-4	Response codes for the endpoint request.
response 1-4 description	Description of the response received for the code.

IV. EXPERIMENT DESIGN

To address our research questions, we developed an experiment setup enabling the comparison between microservice decomposition by human experts and the machine learning-driven approach. Figure 2 illustrates this setup.

Initially, three software engineers established a ground truth, collaboratively developing a set of candidate microservices. This collaboration aimed to minimize the influence of individual biases. The human experts relied on a Swagger visualization of the OpenAPI to scrutinize various attributes, subsequently grouping the services in Excel using their unique operation ID.

For the machine learning approach, our first step involved parsing the API to extract features. These 36 features are detailed in Table II. Some of the API endpoints specify multiple parameters (up to 11) or responses (up to 4). These are set to none if an endpoint has fewer parameters or responses specified.

Each feature underwent a cleaning process to eliminate superfluous whitespaces, special characters, typos, and language errors. This step was crucial, particularly for the subsequent encoding of natural language using word embeddings and term frequency–inverse document frequency (TF-IDF) techniques. We used word embeddings outlined in Table III. In addition to the readily available embeddings, we incorporated two custom embeddings by training the text of two MES domain-specific standards (VDI5600 and IEC 62264) with word2Vec and fastText.

We also varied our encoding approach to investigate potential influences on the decomposition outcome: one involved averaging the encoding into a single feature (see equation 1b for embeddings and equation 2b for TF-IDF), and the other treated each feature independently (see equation 1a for embeddings and equation 2a for TF-IDF).

$$\mathbf{f}_{\text{avg},j} = \frac{1}{m_j} \sum_{i=1}^{m_j} \mathbf{v}_{w_{ji}} \tag{1a}$$

$$\mathbf{f}_{\text{global\_avg}} = \frac{1}{M} \sum_{j=1}^{n} \sum_{i=1}^{m_j} \mathbf{v}_{w_{ij}} \tag{1b}$$

TABLE III
DESCRIPTION OF ENCODINGS USED

Encoding	Description
Combined-fast	Custom 100-dimensional model trained with the fastText algorithm with VDI5600 and IEC62264 standards[a].
Combined-w2v	Custom 100-dimensional model trained with the word2vec algorithm with VDI5600 and IEC62264 standards[a].
crawl-300d-2M-fast	Derived from a 300-dimensional, 2 million word fastText news feed crawl data [11].
enwiki_20180420_100d-w2v	Based on a 100-dimensional word2vec model from English Wikipedia data [12].
text-embedding ada	OpenAI's Ada 1536-dimensional model for advanced natural language understanding [13].
davinci-001	OpenAI's Davinci 12288-dimensional model for deep comprehension and generation capabilities [13].
TF-IDF	Generated using Term Frequency-Inverse Document Frequency to emphasize important words [14].
wiki-news-300d-1M-fast	Created from a 300-dimensional, 1 million word fastText model trained on WikiNews data [11].

[a]learning rate 0.05, window size 5, word occurrence minimum count of 5

$$\text{TF-IDF}_{t,F_j} = \text{TF}(t, F_j) \times \text{IDF}(t) \tag{2a}$$

$$\text{TF-IDF}_t = \text{TF}(t) \times \text{IDF}(t) \tag{2b}$$

We then applied three algorithms for clustering: Affinity Propagation [15], HDBSCAN [16], and k-means [17]. HDBSCAN, like Affinity Propagation, doesn't necessitate presetting the number of clusters and is parameterized by the minimum cluster size. In contrast, k-means requires a predefined number of clusters. We included k-means to explore whether specifying a certain number of clusters would yield better outcomes. For each algorithm a grid search (see Table IV for parameters) was conducted across different combinations of features (path, operation id, summary, tag) and all features – both in averaged and independent settings.

The preference for Affinity Propagation is calculated as follows:

$$d_{ij}^2 = \sum_{k=1}^{d} (x_{ik} - x_{jk})^2 \tag{3}$$

$$s_{ij} = -d_{ij}^2 \tag{4}$$

$$\text{Median}(S), \quad S = \{s_{ij} \,|\, 1 \le i, j \le n, \, i \ne j\} \tag{5}$$

$$\text{Preference} = \text{Median}(S) \times f, \quad f \in \{1, 2, 3, 4\} \tag{6}$$

TABLE IV
CLUSTERING ALGORITHM PARAMETERS

Algorithm	Hyperparameter	Values
Affinity Propagation	preference factor	1, 2, 3, 4
Affinity Propagation	damping	0.5, 0.6, 0.7, 0.8, 0.9
Affinity Propagation	max_iter	100, 200, 300, 400, 500, 600
Affinity Propagation	convergence_iter	10, 20, 30, 40, 50, 60
HDBSCAN	min_cluster_size	2, 3, 4, 5, 6
HDBSCAN	min_samples	None, 5, 10, 15
HDBSCAN	cluster_selection_epsilon	0.0, 0.1, 0.5, 1.0
KMeans	n_clusters	10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
KMeans	init	k-means++, random
KMeans	max_iter	300, 400, 500
KMeans	n_init	auto

Equation 3 calculates the pairwise square euclidean distances. The distance is negated as the Affinity Propagation works on similarities (equation 4). The default preference is then calculated as the median of the result from equation 5. The individual preference for the experiments is then varied by multiplying the default preference with one of the four magnitudes in equation 6.

To evaluate the performance of the processing methods (comprising the algorithm, feature set, encoding, and averaging setting) individually and in comparison to human experts, we employed three distinct metrics: silhouette score [18], Adjusted Rand Index (ARI) [19], and Normalized Mutual Information (NMI) [20]. These metrics were chosen for their ability to gauge the compactness of clusters, the similarity of two assignments, and the mutual information between assignments, adjusted for chance.

V. RESULTS

Our experimental setup involved a comprehensive grid search over feature sets, average settings, encodings, algorithms, and hyperparameters, resulting in 221,696 runs with corresponding cluster (microservice candidates) outcomes. These were tracked using mlflow and analyzed in Python after exporting to CSV files. Concurrently, human experts manually identified 10 microservice candidates for the same MES, leveraging their domain knowledge to establish a benchmark for our automated methods. The largest of these clusters encompassed 94 members, while the smallest contained a single operation and an average size of 29.5 operations. Our analysis, detailed in Figure 3, focuses on the minimum, maximum, and mean values across all runs for each main factor to address each research question.

A. Q1: Performance in a Real-World Scenario Compared to Human Experts

In our study comparing machine learning decomposition to human expert analysis, the maximum values for Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI)

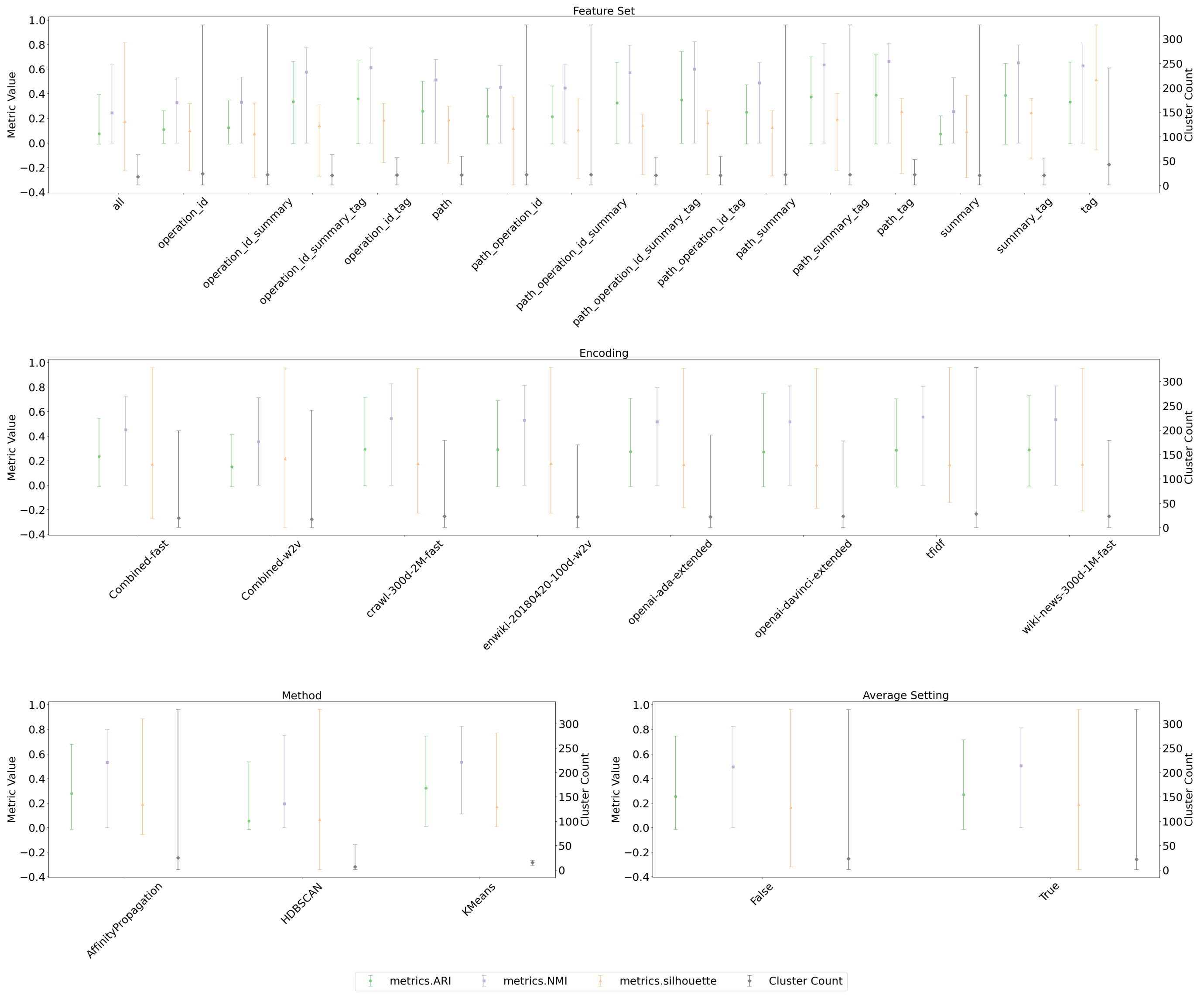


Fig. 3. Comparison of method, average setting, feature set and encoding.

were impressive, reaching 0.745 and 0.825, respectively. This demonstrates the potential of the automated method to closely align with human expertise in optimal scenarios. However, the mean values, standing at 0.321 for ARI and 0.532 for NMI, indicate that achieving such high performance requires careful configuration and tuning of the model to suit specific scenarios.

B. *Q2: Most Effective Attributes in the OpenAPI Specification*

Concerning the effectiveness of various attributes in the OpenAPI specification, our analysis revealed that certain feature combinations consistently led to better decomposition results. Specifically, feature sets that included the 'tag' attribute, particularly when combined with 'path' and 'operation id', showed superior performance in clustering quality. In our real-world example, this combination achieved maximum ARI and NMI scores of 0.717 and 0.814. This suggests that these attributes are key indicators in the OpenAPI specification of our real-world example for identifying cohesive microservice candidates. However, the 'tag' attribute is special to the OpenAPI description in our real-world case as there is no consistent definition of what it should include. The 'path' attribute emerged as the next best alternative (maximum ARI of 0.503 and NMI of 0.679) and performed significantly better than the 'operation id', which was the suggested attribute by [10] (maximum ARI of 0.262 and NMI of 0.530).

C. *Q3: Impact of Feature Encoding on Results*

For the effect of feature encoding on decomposition outcomes, we compared the 'average' (True) and 'non-average'

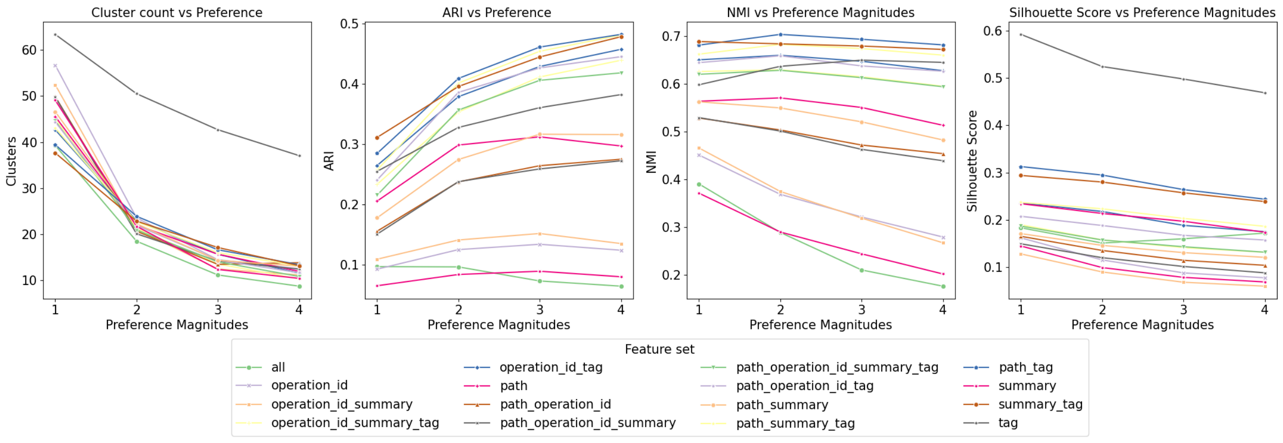


Fig. 4. Influence of preference magnitudes

(False) encoding settings. The results indicate a marginal difference between these two methods: the maximum ARI for 'average' encoding is 0.717, slightly lower than 0.746 for 'non-average' encoding. Similarly, the maximum NMI is 0.814 for 'average' compared to 0.826 for 'non-average.' However, these slight variations are not substantial enough to conclude a definitive preference for one method over the other in the context of our study. Therefore, the choice between these encoding methods can be based more on other factors, such as ease of implementation or computational efficiency.

D. Q4: Enhancement of Results through Domain-Specific Word Embedding

In investigating the effectiveness of encodings, we compared domain-specific custom-trained word embeddings ('Combined-fast' and 'Combined-w2v') but also assessed their performance against other encoding methods, including the simple TF-IDF encoding and more advanced options like the openAI davinci embedding. While our custom embeddings achieved maximum ARIs of 0.546 ('Combined-fast') and 0.412 ('Combined-w2v') and NMIs of 0.726 and 0.714, respectively, they are outperformed by all other encodings. Despite its simplicity, the TF-IDF method reached a notable maximum ARI of 0.704 and NMI of 0.806, demonstrating its effectiveness. However, a significant observation was the high deviation in the mean cluster count (28.29 for TF-IDF) compared to the actual cluster count of 10 identified by human experts. The best result was achieved with the openAI davinci embedding, which recorded the highest maximum ARI of 0.746. This suggests that general embeddings can capture intricate relationships even in very specific domains; thus, training for embeddings on domain-specific document bodies is unnecessary.

E. Q5: Differences in Clustering Algorithms in Terms of Microservice Candidates' Quality

In our comparative analysis of the clustering algorithms, k-means, Affinity Propagation, and HDBSCAN each displayed distinct characteristics in microservice decomposition. k-means demonstrated the most consistent performance with the highest maximum ARI of 0.746 and NMI of 0.826, aligning with expectations due to its use of a predefined number of clusters. Affinity Propagation, with a maximum ARI of 0.679, maximum NMI of 0.797 and a wide-ranging cluster count (1 to 329), resulted in a more varied set of solutions, indicating the need for careful configuration. HDBSCAN, while showing low means of ARIs (0.05) and NMIs (0.19), stood out with a notably high maximum silhouette score of 0.961, suggesting its ability to create well-defined and distinct clusters but with comparable less consistency with the result of the human experts.

VI. Discussion

Our study's application of Affinity Propagation has underscored its potential in microservice decomposition, demonstrating performance that approaches the consistency of the k-means algorithm but with a wider variance, especially in terms of cluster count. The key metrics' progression, as observed across varying preference magnitudes (Figure 4) indicates that while the algorithm efficiently narrows down the candidate pool, it also poses challenges for practical deployment, particularly in new scenarios where ARI and NMI are not viable metrics. The trend in silhouette score presents an interesting contrast. As the preference magnitude increases, the silhouette score decreases. This trend indicates a possible misalignment. With an increasing preference magnitude, the algorithm's suggestions become more similar to the experts' solutions, but the definition and separation of clusters become less distinct. This paradox raises doubts about the algorithm's effectiveness in new environments where the silhouette score is the primary measure available, which may provide limited insight. Thus, even with the use of algorithms, human intervention is still required to identify microservice candidates accurately. Experts must carefully analyze the initial algorithmic propositions and make necessary adjustments to refine the proposed microservice candidates. They may also employ supplementary methods that incorporate further attributes, such as performance and scalability, which go beyond the cohesion focus of the current approach. Despite these considerations, the Affinity Propagation method is still useful in practical problem-solving, as it provides a starting point to identify microservice candidates in a matter of seconds.

VII. Conclusion

This study has successfully demonstrated a method for decomposing a monolithic MES into microservice candidates using OpenAPI specifications, emphasizing the importance of feature selection and algorithm configuration. The alignment of our decomposition results with human expert judgment underscores the potential of the approach. However, there is a noteworthy discrepancy of best runs ARI and NMI metrics that are only available in comparison experiments, with the silhouette score that would be used for evaluating the clustering effectiveness in practical applications. This highlights the essential role of human oversight in semantics-driven microservice decomposition of monolithic applications. Future research should pursue an integrative approach that addresses other microservice attributes, such as scalability and performance, thereby offering a more comprehensive framework for decomposition. Investigating the potential of deep learning clustering approaches, as suggested by [21], may lead to further improvements. Additionally, applying and validating these methods across diverse real-world APIs is needed to solidify the findings and extend the applicability of our study's results.

References

[1] J. Lewis and M. Fowler, "Microservices," 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html

[2] F. Tapia, M. Mora, W. Fuertes, H. Aules, E. Flores, and T. Toulkeridis, "From Monolithic Systems to Microservices: A Comparative Study of Performance," *Applied Sciences*, vol. 10, no. 17, p. 5797, 2020.

[3] F. H. Vera-Rivera, C. Gaona, and H. Astudillo, "Defining and measuring microservice granularity—a literature overview," *PeerJ Computer Science*, vol. 7, p. e695, 2021.

[4] R. A. Schmidt and M. Thiry, "Microservices identification strategies," in *15th Iberian Conference on Information Systems and Technologies (CISTI)*, 2020, pp. 1–6.

[5] M. Kamimura, K. Yano, T. Hatano, and A. Matsuo, "Extracting Candidates of Microservices from Monolithic Application Code," in *25th Asia-Pacific Software Engineering Conference (APSEC)*, 2018, pp. 571–580.

[6] M. Abdullah, W. Iqbal, and A. Erradi, "Unsupervised learning approach for web application auto-decomposition into microservices," *Journal of Systems and Software*, vol. 151, pp. 243–257, 2019.

[7] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, "Service cutter: A systematic approach to service decomposition," in *Service-Oriented and Cloud Computing: 5th IFIP WG 2.14 European Conference*, 2016, pp. 185–200.

[8] L. Baresi, M. Garriga, and A. D. Renzis, "Microservices Identification Through Interface Analysis," in *Service-Oriented and Cloud Computing - 6th IFIP WG 2.14 European Conference, ESOCC 2017*, vol. 10465. Springer, 2017, pp. 19–33.

[9] O. Al-Debagy and P. Martinek, "A New Decomposition Method for Designing Microservices," *Periodica Polytechnica Electrical Engineering and Computer Science*, vol. 63, no. 4, pp. 274–281, 2019.

[10] ——, "Extracting Microservices' Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach," in *IEEE 15th International Conference of System of Systems Engineering (SoSE)*, ser. 2020 IEEE 15th International Conference of System of Systems Engineering (SoSE), 2020, pp. 289–294.

[11] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, "Advances in Pre-Training Distributed Word Representations," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. European Language Resources Association (ELRA), 2018.

[12] I. Yamada, A. Asai, J. Sakuma, H. Shindo, H. Takeda, Y. Takefuji, and Y. Matsumoto, "Wikipedia2Vec: An Efficient Toolkit for Learning and Visualizing the Embeddings of Words and Entities from Wikipedia," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 2020, pp. 23–30.

[13] OpenAI, "Embeddings - OpenAI API," 2023. [Online]. Available: https://platform.openai.com/docs/guides/embeddings/what-are-embeddings

[14] K. Sparck Jones, "A Statistical Interpretation of Term Specificity and Its Application in Retrieval," *Journal of Documentation*, vol. Volume 28, no. Issue 1, pp. 11–21, 1972.

[15] B. J. Frey and D. Dueck, "Clustering by Passing Messages Between Data Points," *Science*, vol. 315, no. 5814, pp. 972–976, 2007.

[16] R. J. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 2013, pp. 160–172.

[17] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[18] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[19] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, vol. 2, no. 1, pp. 193–218, 1985.

[20] A. Lancichinetti, S. Fortunato, and J. Kertész, "Detecting the overlapping and hierarchical community structure in complex networks," *New Journal of Physics*, vol. 11, no. 3, p. 033015, 2009.

[21] M. Ronen, S. E. Finder, and O. Freifeld, "DeepDPM: Deep Clustering With an Unknown Number of Clusters," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, ser. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 9861–9870.