A Demonstration of KAMEL: A Scalable BERT-based System for Trajectory Imputation

Mashaal Musleh
musle005@umn.edu
University of Minnesota, USA

Mohamed F. Mokbel
mokbel@umn.edu
University of Minnesota, USA

ABSTRACT

This demo presents KAMEL; a novel trajectory imputation framework that aims to impute sparse trajectories as a means of increasing their accuracy, and hence the accuracy of their applications. Unlike the large majority of current trajectory imputation techniques, KAMEL does not require the knowledge or the availability of the underlying road network, which makes it applicable to important applications like map inference that need to infer the road network itself. Audience will experience KAMEL through various scenarios that show imputation accuracy as well as KAMEL internals.

ACM Reference Format:
Mashaal Musleh and Mohamed F. Mokbel. 2023. A Demonstration of KAMEL: A Scalable BERT-based System for Trajectory Imputation. In *Companion of the 2023 International Conference on Management of Data (SIGMOD-Companion '23), June 18–23, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3555041.3589733

1 INTRODUCTION

The availability of trajectory data generated from GPS-equipped devices have empowered numerous important applications, including map inference, routing, traffic monitoring, and location-based services. Unfortunately, trajectory data are inherently spare with frequent spatial and temporal gaps, which significantly lower the accuracy of all trajectory-based applications. Hence, several techniques have been proposed for trajectory imputation by trying to predict the trajectory whereabouts within the gaps (e.g., see [2, 6, 16]). Such techniques mainly rely on matching the trajectories to the underlying road network, where the imputation process becomes finding the road network shortest path between each two consecutive trajectory points. Unfortunately, all such techniques have the implicit assumption that the underlying road network is available and reliable, which is not always true [8, 9, 11, 15]. In particular, the whole body of research in trajectory-based map inference applications [1, 10, 14] rely on trajectories to infer the road network. To such applications, one cannot use the road network to impute trajectories, as the road network is not even available, instead, it needs to be inferred. There are only a couple of recent approaches that do not assume road network availability [5, 7]. However, they are only applicable for small road networks, assume the abundance of trajectory data, and can only work offline.

This demo presents KAMEL; a novel trajectory imputation framework that: (a) does not require the knowledge of the underlying road network, (b) scalable to support large geographical areas, and (c) can impute trajectories both offline (bulk mode) and online (streaming trajectories). The main idea of KAMEL is to map the trajectory imputation problem to the *"finding the missing word"* in Natural Language Processing (NLP), which is usually solved using the widely used BERT model [4]. Given a statement like "Paris is the ... of France", where "..." represents a missing word, BERT finds out that the missing word is "capital". KAMEL deals with trajectories as statements, where a trajectory/statement is composed of an ordered set of points/words drawn from a finite pool of possible points/words. Also, points/words in a trajectory/statement are spatially/semantically related and are constrained by rules imposed by the underlying road network/language grammar. Hence, at its core, KAMEL is equipped with a BERT model trained by a set of trajectories, and then used to impute sparse trajectories.

However, using BERT as is within KAMEL results in very poor accuracy [12], mainly due to three main challenges: (1) *Spatial awareness.* BERT is not spatially aware, where it may poorly train its model by including spatially unrelated datasets and/or produce results that are not spatially feasible. (2) *Data ratio.* The original BERT [4] is trained on ~3.3B word corpus composed of ~30K distinct words. Trajectory data is from from this; a typical trajectory dataset would have ~2M GPS points with ~1.5M distinct points. Such low ratio significantly degrades the quality of BERT. (3) *Multiple missing points.* BERT usually aims to find one missing word in a statement, while trajectory imputation may need to find several missing points between two known points. KAMEL overcomes these challenges through its five main components, *Partitioning*, *Spatial Constraints* (both address the *spatial awareness* challenge), *Tokenization*, *Detokenization* (both address the *data ratio* challenge), and *Beam Search* (addresses the *multiple* missing points challenge).

This demo lets conference attendees experience KAMEL in action. conference attendees will be able to upload trajectory data to KAMEL and see how it partitions and trains its models, then use them to impute sparse trajectories accurately. To have an interactive demo, attendees can select different trajectories to impute, inspect imputed points on the map, and compare them with the ground truth and other baseline results. To help attendees understand KAMEL internals, they will be able to change the system parameters to see their effect on the overall accuracy. They will also be able to "debug" KAMEL components through a list of execution steps that they can click on to see their input and output on the map.

This work is supported by NSF under grants IIS-1907855 and IIS-2203553.

SIGMOD-Companion '23, June 18–23, 2023, Seattle, WA, USA

ACM ISBN 978-1-4503-9507-6/23/06…$15.00
https://doi.org/10.1145/3555041.3589733

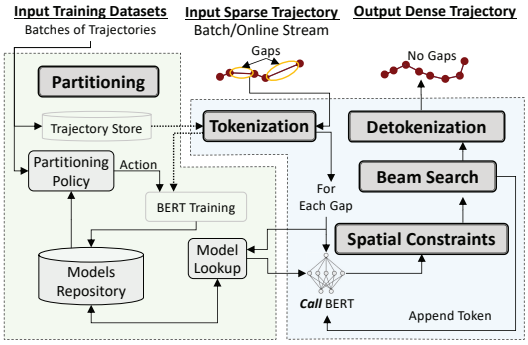

Figure 1: KAMEL **Architecture**

2 KAMEL ARCHITECTURE

Figure 1 depicts the architecture of KAMEL, which is composed of five main components: *Partitioning*, *Tokenization*, *Spatial Constraints*, *Beam Search*, and *Detokenization*. There are two types of inputs to KAMEL: (1) *training data*: new trajectory datasets, which can be fed to the system any time to update or enrich its models. The training datasets go to the *Partitioning* component, represented by the left part of Figure 1, which trains and maintains multiple BERT models in a spatially-aware manner, where each model serves a specific geographical area (details in Section 3). (2) *sparse trajectories*: batches or online stream of trajectories that need to be imputed. The sparse trajectories go to the imputation pipeline, represented by the right part of Figure 1, where each trajectory is processed independently to get its imputed version.

The imputation pipeline starts with the *Tokenization* component (Section 4) which converts each GPS trajectory into a series of tokens. Then, trajectory gaps are identified as two consecutive points apart by a distance larger than a threshold. For each gap, we (1) lookup a model that can serve this area from the *Partitioning* component, and (2) invoke the model to get a list of possible tokens (words) for this gap with their probabilities. These tokens go through the *Spatial Constraints* component (Section 5) which injects spatial awareness into the results by rejecting any token that is spatially infeasible. Accepted tokens then go to the *Beam Search* component (Section 6) which is responsible for finding a *sequence* of tokens that fill the gap with the highest probability. This sequence goes to the last component in the pipeline, the *Detokenization* (Section 7) which accurately converts each token into a GPS point, and finally returns the output dense trajectory.

3 PARTITIONING

Represented by the left side of Figure 1, the *Partitioning* component is responsible for the injecting spatial awareness in the training process of BERT model within KAMEL. It receives the input batches of training data and partitions them into spatially related parts to ensure BERT learns properly and achieves good accuracy. It continuously maintains models by partitioning and updating them as needed when new related data arrive. Each partition is used to train a model that serves a specific geographical region. This component consists of the following two modules:

Partitioning Policy. This module receives batches of trajectories for training. For each batch, it decides to take one of three possible actions: (1) *Create New Model*: when trajectories cover a completely new geographic area that KAMEL has no prior model for, (2) *Enrich Existing Model*, when trajectories cover an area that significantly overlaps with one of the current areas that KAMEL already has a model for, (3) *Merge Existing Models*, when trajectories cover an area that is kind of adjacent (or has a little overlap) to one of the existing areas that KAMEL has a model for.

Models Repository. This module maintains all trained models in disk, indexed by a QuadTree for efficient access. It executes spatial queries to retrieve trained models based on a query bounding box.

4 TOKENIZATION

One of the challenges of using BERT in KAMEL is the very small ratio of the number of distinct GPS points to the number of all points in a trajectory data set, compared to the ratio of the number of distinct words in a word corpus that BERT is used to deal with. To address this challenge, the *Tokenization* component in Figure 1 partitions the space into small cells, where each cell is considered as a token that represents all points within it. Then, each input trajectory is presented as a sequence of cells instead of a sequence of points. Hence, BERT can be fed with a set of distinct cells instead of distinct points. This significantly raises the ratio of distinct tokens in a dataset, which addresses the small ratio challenge. To this end, the *Tokenization* component employs the following two techniques:

Hexagonal Cells. For its tokenization scheme, KAMEL uses a flat *hexagonal* grid structure based on Uber's H3 Hexagonal Hierarchical Spatial Index [3]. Basically, the whole world geographical area is divided into a set of non overlapping hexagons. The rationale of using hexagons over the traditional square or rectangular partitioning is: (1) A hexagonal grid would provide a regular neighborhood structure, where all neighboring cells have identical features in terms of centroid distances and shared edges between them, which is more suitable for the case of predicting transitions between cells, and hence trajectory imputation, (2) Most applications that use rectangular grids mainly do so due to its hierarchical structure, where it is not as simple to make a hierarchy of hexagons. However, in the imputation process, we really do not need a hierarchy, and hence we do not have to be tied to rectangular cells, and (3) The cost of mapping a point to its hexagonal cell has the same order of mapping a point to a rectangular cell, so, we are not sacrificing efficiency here while using a hexagonal grid.

Cell Size. The cell size achieves an unusual tradeoff between the total number of tokens and the precision of imputed trajectories. Large cell size means fewer distinct tokens, and hence a better ability to employ BERT. Yet, given the cell size is large, the imputed cells would not well represent actual trajectories. This may call for having small cells. However, very small cell size highly increases the number of distinct tokens, which may not work well with BERT. The *Tokenizaiton* module employs a cell size selection strategy to decide on the right cell size for newly incoming training datasets.

5 SPATIAL CONSTRAINTS

As BERT is not really spatially aware, it may end up in producing imputed trajectories that are not spatially feasible. Hence, the *Spatial Constraints* component is responsible for injecting spatial awareness in the output part of BERT, before passing it to the next

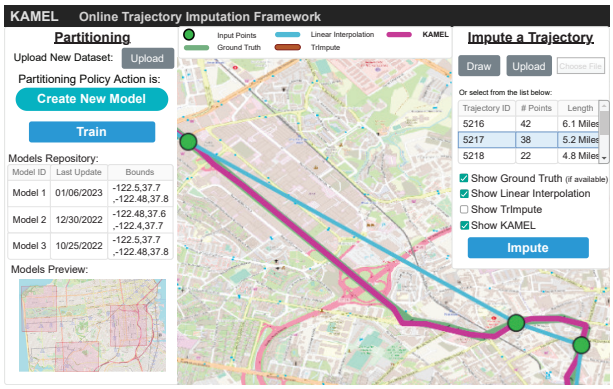

Figure 2: Scenario 1: Upload Data and Impute Trajectories

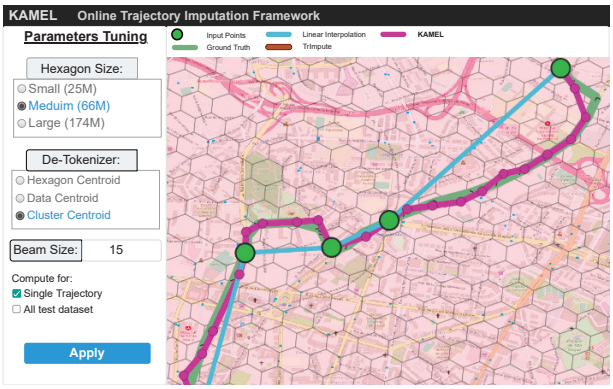

Figure 3: Scenario 2A: Parameters Tuning (Single Trajectory)

component (*Beam Search*). It basically receives BERT output as a set of candidate imputed points, and rejects some of them that do not satisfy a list of preset spatial constraints, including: (1) to avoid cycles in the imputed trajectory, a token should not appear more than once in the imputed path, and (2) An imputed token should be within a threshold of the spatial proximity of both endpoints of the imputed segment. The threshold is computed based on the segment temporal length, taking into account that a vehicle can only travel a limited distance for a given period.

6 BEAM SEARCH

While BERT is designed to find one missing word, which is equivalent to find one missing point in trajectories, an imputed trajectory would need to find multiple points between each two consecutive points. The *Beam Search* component addresses the multiple points challenge by computing the joint probability for a sequence of multiple points together and searches for the sequence with the highest probability. To do so, the *Beam Search* component receives the output tokens from the *Spatial Constraints* component and iteratively calls BERT (i.e., appending the received output to the input of the next call) to receive new candidate tokens, and so on to form a sequence of imputed tokens. It computes their joint probability by multiplying the probability of each token, which is provided by BERT after each call. Beam search is closely similar to breadth first search, but keeps only the top B results instead of all results. B is a parameter as a trade-off between accuracy and computational cost.

7 DETOKENIZATION

The output of the *Beam Search* component is imputed trajectories, where each trajectory is presented as a sequence of tokens (hexagonal cell IDs) instead of a sequence of points. The *Detokenization* is responsible for converting this back to a sequence of points and sending it directly to the users as KAMEL output. The *Detokenization* component employs the following three options for its process:

Cluster Centroid. Form several clusters from training points within a cell based on their travel direction and then use the centroid of the most likely cluster. The rationale for this is that when a cell covers multiple roads or an intersection, points are highly likely to have different directions. Choosing the target point this way makes it close to the road that has a similar direction.

Data Centroid. If there is not enough data to form and support different clusters, we use the centroid of all the points in the cell as one cluster. This is because most cell space is likely to be empty except for only a few bands or lines of points along the roads covered by this cell. Taking the centroid of these points helps in making the target point somewhere close to these few roads.

Cell Centroid. If there is absolutely no enough data to form clusters, we fall back to picking the centroid of the cell area.

8 DEMO SCENARIOS

This section shows several demo scenarios where the conference attendees can interact with KAMEL to understand its operations from the point of view of: (a) Regular users who just need to use the system to impute their sparse trajectories, and (b) Researchers, developers, and practitioners who want to understand the system internals. Since model training is an offline process that takes significant time, we will provide pretrained models for several trajectory datasets in advance to ensure interactivity. Then, the demo will be mainly based on Porto Dataset [13] with 1.7M trajectories for real taxi trips in the city of Porto, Portugal, driven for a total length of ~8.8M km, with ~83M GPS points spanning an area of ~500 km^2. However, conference participants can use any trajectory data they would like, especially for the online imputation, which they can upload in CSV format (trajectory ID, latitude, longitude, and timestamp), or even draw as sparse trajectories on the map.

8.1 Scenario 1: Trajectory Imputation

Figure 2 depicts the user interface of this scenario, which lets demo attendees experience the actual usage of KAMEL: train it and then use it to impute sparse trajectories. On the left panel, attendees can see the list of models currently maintained by KAMEL along with a small map preview of their partitions. Attendees can also feed KAMEL with additional training trajectories (data and pre-trained models will be provided for interactivity) and see how it adjusts its partitions, accordingly. For example, when KAMEL has a trained model for Minneapolis city and the uploaded data contain new additional trajectories in the same city, then attendees will see the chosen action as *Enrich Existing Model*. Yet, if the uploaded data is for Saint Paul, which is an adjacent city to Minneapolis, then the partitioning action will be the *Merge Existing Models*.

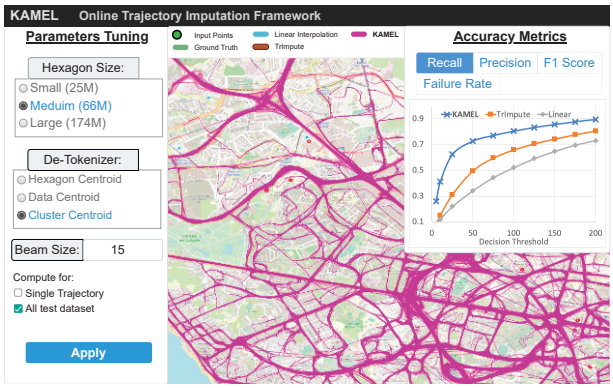

Figure 4: Scenario 2B: Parameters Tuning (Dataset)

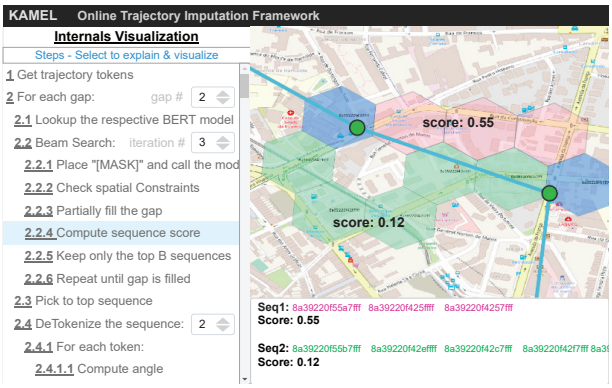

Figure 5: Scenario 3: System Internals

On the right panel of Figure 2, attendees can provide trajectories to impute either by selecting from the list of available trajectories, uploading new test trajectories, or creating a new one by plotting sparse points on the map. Attendees will see the imputed trajectory plotted on the main map in the center, and can inspect its newly added points. They can also decide to override the default model that matches the trajectory geographic area with another model. This will help to see how the accuracy can be degraded if a wrong model is picked up for imputation. The output of different baselines will also be shown on the map in different colors as a means of comparison with KAMEL and ground truth (if available).

8.2 Scenario 2: Parameters Tuning

This scenario aims to help demo participants understand the impact of various parameters on the accuracy of the trajectory imputation process in KAMEL, for both a single trajectory (Figure 3) and a set of trajectories (Figure 4). On the left panel, for both cases, participants can change KAMEL parameters, hexagon size, beam size, and the detokenizer options, and check the imputation result accordingly, on the main map. Participants can experiment with increasing the beam size and/or cell size until finding optimal values at which the accuracy does not change significantly. For the case of a trajectory dataset (Figure 4), participants can also get various accuracy metrics computed for the whole dataset, plotted in a chart on the right panel. This includes recall, precision, F1 score, and failure Rate, which would help participants quantify the effect parameters tuning.

8.3 Scenario 3: Visualize Internal Execution

This scenario is mainly for system researchers who are interested in knowing more about the system kernel works. Figure 5 depicts the user interface of this scenario, where the left panel breaks down the imputation procedure into a series of steps, including: "Get Trajectory Tokens", "Look Up a Corresponding BERT Model", "Invoke BERT Model", "Check Spatial Constraints", "Append Token", "Compute the Sequence Joint Probability", "Detokenize the Sequence". Demo participants can click on any step to visualize it in the main map and see its inputs, outputs, and all its other related information. For example, the demo audience can click on the "Get Trajectory Tokens" step to see the input GPS points and their respective hexagons highlighted on the map, or click on the "Invoke BERT Model" step to see the input that is being sent to BERT, how it is formatted, where the gap is placed between the input (the missing part that BERT is asked to predict), and the returned tokens from BERT and their probabilities, all highlighted on the map and described on the info box beneath it. This is similar to looking at a query pipeline and seeing the intermediate results between query operators.

For steps that repeat (i.e., loop) such as "Beam Search" or "Invoke BERT Model", there is a numerical counter next to such steps where participants can increase/decrease to execute the next/previous iteration, or directly type in the number (i.e., 2 or 3) to indicate the iteration they would like to visualize. This explanation or debugging style allows the conference audience to greatly understand and comprehend the ideas behind KAMEL and how its components integrate with each other to provide accurate imputation results.

REFERENCES

[1] S. Abbar, M. Alizadeh, F. Bastani, S. Chawla, S. He, H. Balakrishnan, and S. Madden. The Science of Algorithmic Map Inference (Tutorial). In *KDD*, 2018.
[2] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On Map-Matching Vehicle Tracking Data. In *VLDB*, 2005.
[3] I. Brodsky. H3: Uber's Hexagonal Hierarchical Spatial Index. https://eng.uber.com/h3/.
[4] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: Pre-training Deep Bidirectional Transformers for Language Understanding. *CoRR*, abs/1810.04805, 2018.
[5] M. M. Elshrif, K. Isufaj, and M. F. Mokbel. Network-less trajectory imputation. In *SIGSPATIAL*, 2022.
[6] C. Guo, B. Yang, J. Hu, and C. Jensen. Learning to Route with Sparse Trajectory Sets. In *ICDE*, 2018.
[7] Y. Li, Y. Li, D. Gunopulos, and L. J. Guibas. Knowledge-based Trajectory Completion from Sparse GPS Samples. In *SIGSPATIAL*, 2016.
[8] Mapillary. Unveiling the Mapping in Logistics Report: The Impact of Broken Maps on Last-Mile Deliveries. https://blog.mapillary.com/update/2020/02/14/mapping-in-logistics.html.
[9] Discover New Roads with Bing Maps. https://blogs.bing.com/maps/2022-12/Bing-Maps-is-bringing-new-roads/.
[10] M. Musleh, S. Abbar, R. Stanojevic, and M. F. Mokbel. QARTA: An ML-based System for Accurate Map Services. *PVLDB*, 14(11), 2021.
[11] M. Musleh and M. F. Mokbel. RASED: A Scalable Dashboard for Monitoring Road Network Updates in OSM. In *MDM*, 2022.
[12] M. Musleh, M. F. Mokbel, and S. Abbar. Let's Speak Trajectories. In *SIGSPATIAL*, pages 37:1–37:4, Seattle, WA, USA, Nov. 2022.
[13] Taxi Service Trajectory. Prediction Challenge. ECML PKDD 2015. http://www.geolink.pt/ecmlpkdd2015-challenge/dataset.html.
[14] S. Ruan, C. Long, J. Bao, C. Li, Z. Yu, R. Li, Y. Liang, T. He, and Y. Zheng. Learning to Generate Maps from Trajectories. In *AAAI*, 2020.
[15] Traffic Technology Today. Poor maps costing delivery companies US $6bn annually. https://www.traffictechnologytoday.com/news/mapping/poor-maps-costing-delivery-companies-us6bn-annually.html.
[16] K. Zheng, Y. Zheng, X. Xie, and X. Zhou. Reducing Uncertainty of Low-Sampling-Rate Trajectories. In *ICDE*, 2012.