Mobility-Aware Cache Framework for Scalable LLM-Based Human Mobility Simulation

Hua Yan
Lehigh University
Bethlehem, USA
huy222@lehigh.edu

Heng Tan
Lehigh University
Bethlehem, USA
het221@lehigh.edu

Yingxue Zhang
State University of New York at Binghamton
Binghamton, USA
yzhang42@binghamton.edu

Yu Yang
Lehigh University
Bethlehem, USA
yuyang@lehigh.edu

Abstract

Simulating large-scale human mobility is fundamental to understanding population movement patterns and supporting real-world geospatial applications such as urban planning, epidemic response, and transportation analysis. Recent works treat large language models (LLMs) as human agents to simulate realistic mobility behaviors using structured reasoning, but their high computational cost limits scalability. To address this, we design a mobility-aware cache framework named MobCache that leverages reconstructible caches to enable efficient large-scale human mobility simulations. It consists of: (1) a reasoning component that encodes each reasoning step as a latent-space embedding and uses a latent-space evaluator to enable the reuse and recombination of reasoning steps; and (2) a decoding component that employs a lightweight decoder trained with mobility law-constrained distillation to translate latent-space reasoning chains into natural language, thereby improving simulation efficiency while maintaining fidelity. Experiments show that MobCache significantly improves efficiency across multiple dimensions while maintaining performance comparable to state-of-the-art LLM-based methods.

CCS Concepts

• **Computing methodologies** → **Simulation evaluation**.

Keywords

Mobility simulation; Large language model; Large-scale simulation

1 Introduction

Human mobility modeling plays a fundamental role in supporting a wide range of downstream geospatial and spatiotemporal applications, including urban planning [45, 46], epidemiology [11, 14], and transportation analysis [32, 40]. Effective human mobility analysis typically relies on large-scale, fine-grained mobility data that capture population-level movement patterns over space and time. In practice, such data are mainly obtained through two channels. Travel surveys record individual trips but suffer from recall bias, sparse temporal sampling, and high collection costs [4, 42]. Sensor-based tracking, such as mobile-phone traces or Bluetooth beacons, provides denser temporal coverage but depends on device penetration and raises significant privacy concerns [7, 23]. As a result, both channels face challenges in scaling to millions of agents while preserving privacy, motivating the need for privacy-preserving alternatives such as mobility simulation.

A growing body of recent work leverages large language models (LLMs) to simulate human mobility trajectories without relying on real mobility traces [1, 10, 19, 21, 25, 29, 34, 38], which show promising results. These methods typically model an LLM as a virtual human agent and prompt it to perform step-by-step reasoning over mobility intentions and activities. While such methods can produce realistic mobility behaviors, they often incur substantial computational costs. For instance, simulating one million agents for a single day can exceed $1,000 in API fees under current token pricing schemes [6, 37]. There are two main methods to reduce costs. First, a group-based methods divides agents into coarse profile groups (e.g., by job and income) and calls the LLM once per group to generate trajectories [8]. However, the method forces every agent in the same group to share identical behavior, which reduces individual diversity. Second, by processing multiple I/O operations simultaneously and reusing TCP connections, the system can handle many requests at the same time [34, 48]. But each agent still needs one LLM call for every trajectory, so the total monetary cost of these methods remains high.

A natural way to reduce repeated LLM calls is *response caching*, where input–output query pairs are stored locally and reused whenever a similar query arises. This caching paradigm has proven effective in many domains such as e-commence [52], intelligent QA systems [9] and Machine Translation [15]. Note that this paradigm is orthogonal to other caching mechanisms such as KV caching [20, 54], and we provide a comparison in the related work section. However, response caching suppresses diversity. Common queries, such as *"What is the next activity for a software engineer who has just finished work?"*, often hit the same cached answer, for example a nine-to-five schedule, even though real workers also take night shifts, overtime, or flexible hours. The resulting lack of behavioral variation lowers simulation fidelity (see Section 2.1 and Section 4.5).

To overcome this limitation, we design a new caching paradigm: rather than caching an LLM's final responses, we cache the intermediate reasoning steps it produces while generating mobility data. This design builds on prior work [19, 38] showing that mobility behaviors are typically produced through multi-step reasoning rather than a single step. We represent these reasoning steps as nodes in a tree, so that cached steps can be recombined into new reasoning chains (Figure 1). This recombination lets a single cached step serve

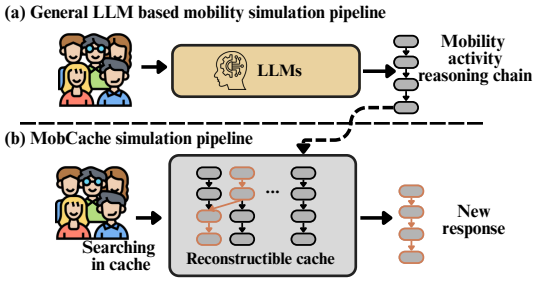

Figure 1: Core idea of MobCache.

many responses, improving both reuse and diversity. We call the resulting approach *reconstructible caches*. A straightforward implementation would keep each step as natural-language text and build new chains by concatenating steps from different cached chains. This increases diversity, but it cannot ensure mobility awareness, such as the spatial and temporal constraints inherent to human movement [44, 51], and therefore limits fidelity.

To this end, we design a mobility-aware cache framework that builds reconstructible caches to enable efficient large-scale human mobility simulations. The framework makes mobility-aware reasoning chains reusable and recombinable by addressing two challenges, one in reasoning and one in decoding:

(1) Reasoning perspective: Reasoning chains are hard to make mobility-aware when each step is an explicit language token, since such tokens cannot flexibly encode mobility-specific constraints. We therefore move the reasoning process into a latent space [18, 41], representing each step as an embedding rather than as language tokens. This brings two benefits. First, mobility constraints can be written directly into the embeddings during training. Second, the embeddings implicitly capture diverse reasoning patterns and decision paths [18, 36], so that logically consistent reasoning chains can be explored efficiently during mobility-aware decoding (see below). We also train a latent-space evaluator, guided by a fine-tuned LLM, to identify valid reasoning paths.

(2) Decoding perspective: After obtaining the reconstructed reasoning chains in the latent space, we design a lightweight decoder trained through mobility law-constrained distillation. This decoder efficiently translates latent reasoning chains into natural language. This strategy avoids repeated calls to the original LLMs for decoding while achieving comparable performance, thereby ensuring efficiency and fidelity in large-scale simulations.

In particular, our main contributions are as follows.

- We introduce building reconstructible caches in latent space to accelerate large-scale LLM-based mobility simulation, aiming to improve both simulation efficiency and the diversity of the generated mobility data.
- We present MobCache, a mobility-aware cache framework with two key components: (1) a reconstructible cache that stores latent-space reasoning embeddings from a fine-tuned LLM and supports tree-structured search, so cached reasoning can be flexibly recombined; and (2) a lightweight decoder, trained via mobility law-constrained distillation, that converts latent-space reasoning chains into natural language while preserving spatial and temporal consistency.
- We conduct extensive experiments showing that MobCache outperforms all baselines on efficiency metrics while staying comparable on quality metrics. It reduces inference time by at least 83.61%, increases tokens per second by 53.26%, and lowers cost by 71.82%. In a case study, applying MobCache to a state-of-the-art simulation baseline reduces inference time by 65.49% and cost by 40.71% with no loss in quality.

2 Motivation

2.1 Why cache diversity matters

We run a simple experiment to show that a cache with limited diversity degrades simulation quality. From a real-world dataset (Section 4), we randomly sample 10,000 trajectories as ground truth. We then simulate trajectories with LLM-archetypes [8], a common group-based method. It groups people into clusters by profile attributes such as occupation and income, and keeps only a small set of real trajectories, 1,000 across all clusters. To generate 10,000 new trajectories, we assign 10,000 new users to the most similar cluster by profile and samples one stored trajectory from that cluster. When the store is small, many users receive the same trajectory, which limits diversity.

We compare the ground-truth and simulated sets on two standard mobility signals. (1) **Stay duration.** For each trajectory, we compute the mean duration of all stops, then compare the two distributions. (2) **Location coverage.** We divide the study area into 1 km×1 km grids, identify the 15 grids visited most often in the ground-truth set, and count how often the baseline visits those same grids.

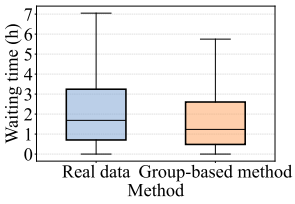

Figure 2: Comparison of stay duration distributions.

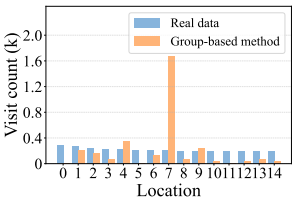

Figure 3: Comparison of location coverage.

Figure 2 compares the stay-duration distributions. Compared to the real data, the group-based method produces stay durations that are more tightly concentrated, indicating less behavioral diversity. Figure 3 illustrates the location coverage. The results show that the group-based approach underrepresents many of the most frequently visited locations in the real data. Together, these results show that limited cache diversity reduces the realism of mobility simulation, which motivates a more flexible cache that can generalize beyond the stored samples.

2.2 Logic inconsistency of reasoning chain reconstruction in language space

A natural way to reuse cached reasoning chains is to recombine their steps directly in language space, that is, as natural-language text. Suppose the cache holds many textual reasoning chains. While following one chain, we may want to branch at some step and continue it with a plausible step drawn from another chain (as shown in Figure 1). A simple way to find such a step is to embed the current step and the candidate steps with a language model (e.g., `sentence-transformers/all-MiniLM-L6-v2`) and link two

steps whenever their similarity exceeds a threshold, forming a new chain.

However, recombining steps this way has a key limitation: the text can read fluently while the underlying logic is wrong. Box 2.2 shows such a case. The third step describes shopping for an *upcoming holiday* on November 2nd, although no holiday falls near that date. It also measures the shopping distance from the person's *workplace*, yet the previous step left her at home and the day is a weekend, so she would not be at work. The recombined steps are therefore logically inconsistent.

Bad example

This person is a 28-year-old woman. The education level is High school diploma, and consumption level is medium. Home and Workplace location: `(116.000, 40.000)`, ...; Date: `2019-11-02`

- At 12:30 a.m., the woman returns home after a late-night social outing with friends, having spent the evening at a nearby restaurant. The distance traveled is approximately 2 km.
- At 10:00 a.m., she wakes up and decides to go for a morning walk in a nearby park to enjoy the fresh air. The park is about 1 km from her home.
- At 4:00 p.m., she heads to a shopping area to prepare for **upcoming holiday events**. The shopping area is about **3 km from her workplace**.

3 Design

3.1 Problem formulation

Given individual profiles (age, occupation, and home locations) and the city's points of interest (POIs), our goal is to design an efficient human mobility simulator. Taking the profiles and POIs as input, the simulator outputs a daily mobility trajectory for each individual. We represent a trajectory as a sequence of event-level spatiotemporal points rather than a continuously sampled GPS trace; each point is an activity event with a timestamp and latitude/longitude coordinates. The simulator should produce realistic trajectories while scaling to tens or hundreds of thousands of individuals.

3.2 Overview

We design MobCache, a mobility-aware caching framework that builds reconstructible caches for efficient, large-scale human mobility simulation. Figure 4 shows its workflow, which has two phases: a **training phase** that builds reconstructible caches from LLM-generated trajectories, and an **inference phase** that generates new trajectories by reusing the caches.

Training phase: We first use an existing LLM (e.g., GPT) with task-specific prompts to generate a small-scale mobility dataset; the same data can also be drawn from prior mobility simulators [38]. Each example pairs a **reasoning chain** with the resulting **mobility activities**, both in text, where the reasoning chain captures the intention behind each activity. We then fine-tune the LLM on this data with a latent-space reasoning strategy [18] so that it reasons in latent space: each example now pairs a **latent-space reasoning chain** with the textual mobility activities. We use these latent-space chains to build a reconstructible cache, whose construction has three parts: (1) storage of latent-space reasoning chains, which supports tree-structured search for flexible reconstruction; (2) a latent-space evaluator, which judges whether a new branch added to a chain is plausible; and (3) a lightweight decoder, which turns latent-space reasoning chains back into textual mobility activities.

Inference phase: For a new user, we first identify the cached users whose context (e.g., profile and date) is most similar. Starting from a matched user, we obtain a latent-space reasoning chain for the new user, either by following an existing chain or by exploring the cache to build a new one. We then decode the chain into textual mobility activities with the lightweight decoder and map those activities to real geographic locations with a mapping model.

3.3 LLMs fine-tuning for latent-space reasoning

3.3.1 ***Data initialization.*** We first leverage general LLM-based human mobility simulation methods to generate a small-scale mobility dataset for cache construction. In our work, we use GPT-4o-mini as the generator with task-specific prompts, but this step can be replaced by other LLM-API-based human mobility simulation method [1, 38]. Following prior work, each generated example is a text pair of a reasoning chain and its mobility activities, where the reasoning chain captures the intention behind each activity. To ensure this format, we construct prompts that include the person's demographic profile (e.g., income and occupation), the date, nearby POIs around their home and workplace, and other task-specific requirements. An example is provided in the appendix.

3.3.2 ***Latent-space reasoning.*** To make reasoning chains reusable, we move the reasoning process into the latent space [18, 41], where each reasoning step is represented as a latent-space embedding instead of a language token. In standard chain-of-thought (CoT) reasoning training, the LLM is supervised to output both intermediate reasoning steps and the final answer as text. In our latent-space reasoning training, we replaces the explicit reasoning step outputs with latent-space reasoning embeddings, following existing work [18]. Specifically, given an input prompt, the LLM is trained to produce latent-space reasoning embeddings r_t at each reasoning step, which replace token-level outputs and are recursively fed back into the LLM as inputs for subsequent reasoning, while the final answer (i.e., the mobility activities) is still generated as text. The process is formulated as $r_t = f_\theta(q, R_{1:t-1})$, where q is the input prompt and $R_{1:t-1} = [r_1, r_2, \ldots, r_{t-1}]$ are previous latent-space reasoning embeddings. The latent-space reasoning embedding r_t is used as input for generating the next latent-space reasoning embedding.

Multi-stage training process: We achieve the transition from CoT reasoning to latent-space reasoning through multi-stage training. The LLM is first fine-tuned using standard CoT training. We then replace the textual reasoning steps with latent embeddings one stage at a time, from the earliest step to the latest. Specifically, at each stage t, the first t reasoning steps are replaced by latent embeddings $R_{1:t}$, while the remaining reasoning steps and the final answer are still supervised in text. This continues until all reasoning is performed in the latent space, while the final answer y (i.e., mobility activities) remains under language-level supervision: $\max_\theta \log p_\theta(y \mid q, R_{1:t})$.

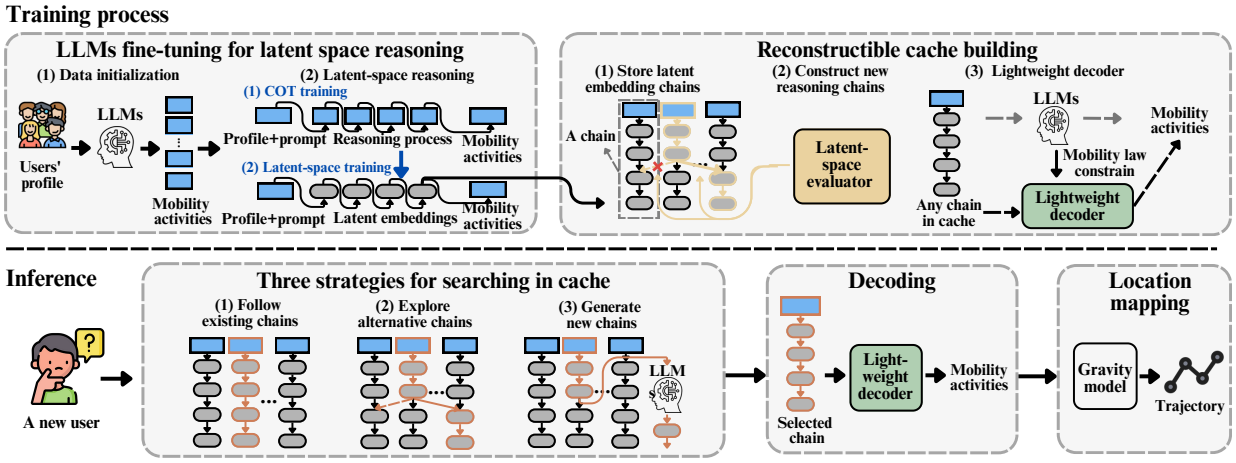

Figure 4: Framework of MobCache.

3.4 Reconstructible cache building

3.4.1 ***Store latent-space reasoning chains.*** Each cache entry stores one chain, consisting of (1) the **prompt** q, the initial task input; and (2) the **latent-space reasoning embeddings** $R_{1:t} = [r_1, r_2, \ldots, r_t]$, the intermediate reasoning steps, where r_t is the embedding at step t. Treating each embedding as a node, the cached chains together form a tree: a branch can extend from any intermediate node of one chain to nodes in other chains. This shared structure is what enables tree-structured search and reasoning recombination.

3.4.2 ***Construct new reasoning chains.*** After storing the latent-space reasoning chains in the cache, we can further utilize these cached reasoning chains to generate new reasoning chains by branching from existing chains. However, latent-space reasoning embeddings are not interpretable by humans, making it hard to determine which branches are logically valid and consistent with human reasoning.

We therefore train a latent-space evaluator that scores how promising a branch is. Formally, the latent-space evaluator model g_ξ is trained to predict a score: $\sigma_t = g_\xi(q, R_{1:t-1}, r_t)$, where q is the prompt and $R_{1:t-1} = [r_1, r_2, \ldots, r_{t-1}]$ is the sequence of previous reasoning steps. σ_t indicates the estimated quality of including r_t as the next reasoning step. We implement the evaluator with a Transformer, which captures the contextual dependencies within a chain when judging coherence.

Label construction: To construct labels for latent-space evaluator training, we adopt a similarity-based method: **(1) Generate next latent-space reasoning step.** For each training example $(q, R_{1:t-1})$, we use the fine-tuned LLM (details in Section. 3.3) to generate the next latent-space reasoning steps: $\hat{r}_t$. **(2) Compute similarity-based labels.** Given a candidate reasoning r_t, its supervision label b_t is computed as: $b_t = \mathrm{sim}(r_t, \hat{r}_t)$, where $\mathrm{sim}(\cdot,\cdot)$ is a similarity function (e.g., embedding cosine similarity or model-based scoring). Intuitively, b_t is high if r_t is similar to next latent-space reasoning steps: $\hat{r}_t$, and low otherwise. The evaluator is trained to minimize the squared error between the predicted scores and supervision labels $\mathcal{L} = \sum_t (\sigma_t - b_t)^2$.

3.4.3 ***Lightweight decoder.*** Finally, we need to convert the cached latent-space reasoning chains into textual activities. The LLM from Section 3.3 can do this, but relying on it for every decode is expensive, so we train a small decoder to take its place.

Distillation: Given a reasoning embedding chain $R = (r_1, \ldots, r_T)$, the decoder generates an activity sequence $Y = (y_1, \ldots, y_L)$ autoregressively, where r_t is the embedding of the t-th reasoning step and y_l is the l-th output token. We treat the original LLM as the **teacher decoder** and train a **lightweight student decoder** to reproduce the same mapping from R to Y, using cross-entropy loss: $\mathcal{L}_{\text{distill}} = -\sum_{l=1}^{L} \log P_\rho\left(y_l \mid y_{<l}, \hat{R}\right)$, where ρ are the student decoder's parameters, y_l is the l-th token produced by the teacher, and $\hat{R}$ is the student-side input. Because the teacher's embeddings R and the student differ in dimensionality, we add an MLP projector e_μ that maps each teacher embedding into the student's space, giving $\hat{R} = e_\mu(R)$ as the decoder input.

Mobility law constraint: To improve distillation quality, we add explicit mobility-law constraints. The idea is to penalize differences between the mobility-law statistics of the student's outputs and the teacher's. From the teacher's activity sequences Y_{teacher}, a statistics function $d(\cdot)$ extracts mobility-law features (e.g., jump distances), giving a target distribution $p_{\text{teacher}}(d(Y))$. To keep the student side differentiable, we learn a function z_τ (e.g., an MLP) that predicts the mobility-law distribution from the student's hidden states h_{light}. We then push $z_\tau(h_{\text{light}})$ toward the teacher distribution by minimizing the Kullback–Leibler (KL) divergence: $\mathcal{L}_{\text{law}} = \mathrm{KL}\big(z_\tau(h_{\text{light}}) \,\|\, p_{\text{teacher}}(d(Y))\big)$.

Training objective: The training objective combines the distillation loss L_{distill} with the mobility law constraint loss $L_{\text{total}} = L_{\text{distill}} + \lambda L_{\text{law}}$, where λ is a trade-off parameter.

3.5 Inference

To generate a trajectory for a new user, we first find the cached users whose context (e.g., profile and date) is most similar. If no cached user passes a predefined similarity threshold, we call the fine-tuned LLM to generate a fresh reasoning chain and its activities, and store them in the cache for reuse. If a similar cached user exists, we search the cache for a reasoning chain using one of the following strategies:

- **Follow existing chains.** We directly use the reasoning chain of the most similar cached user for the new user.

- **Explore alternative chains.** Starting from the reasoning chain of the retrieved similar user, we go through the chain and randomly select a node as a branching point. From this point, we search the cache for several candidate latent-space embedding nodes from other reasoning chains based on embedding similarity. A latent-space evaluator then scores these candidates and selects the most plausible one. The selected node is appended to the current chain, after which we repeat the above procedure by selecting a new branching point and extending the chain again. This process is repeated for a fixed number of iterations.
- **Generate new chains.** If the latent-space evaluator assigns low scores to all available branching nodes, we invoke the LLM to generate a new reasoning chain, which is then stored in the cache for future reuse.

We use an exploration rate to control the probability that the model follows an existing chain or explores a new branch. After obtaining a valid reasoning chain using these strategies, we input it into the lightweight decoder to decode the corresponding textual mobility activities. Finally, these textual activities are mapped to real geographic locations to form trajectories using the gravity model.

4 Evaluation

4.1 Dataset description

We use two public mobility datasets. One dataset was collected in Beijing, China, covering the period from October 1, 2019 to December 31, 2019, and includes mobility records for 100,000 individuals [38]. The dataset contains mobility trajectories and user profile information (e.g., age, gender, and occupation), collected via a social networking platform. Another dataset is the NYC POI check-in dataset [50]. For this dataset, we simulate users' profiles based on U.S. Census demographics [5].

4.2 Evaluation setup

4.2.1 ***Evaluation configuration.*** For the Beijing dataset, we preprocess the raw mobility records by removing users with incomplete profiles and extracting stay points using a 20-minute, 500-meter threshold following prior work [16, 39]. From the preprocessed dataset, we select approximately 1,000 users and use an LLM to generate 13,000 synthetic human mobility trajectories based on their profiles. We fine-tune an LLM on these synthetic trajectories (Section 3.3) so that it reasons in latent space, and store the resulting latent-space reasoning chains as our mobility cache. For evaluation, we sample a separate set of 20,000 real trajectories as the test set, with no user overlap with the cache, and compare the generated trajectories against it.

For the NYC POI check-in dataset, we select approximately 400 users and use an LLM to generate two weeks of synthetic mobility trajectories based on their profiles. We fine-tune an LLM on these synthetic trajectories for mobility cache building (Section 3.3). For evaluation, we construct a separate test set from six months of real check-in trajectories, containing approximately 16,000 trajectories.

4.2.2 ***Implementation.*** We implement our method using Python 3.10 and PyTorch 2.1.0. **(1) Latent-space reasoning.** we fine-tune a LLaMA-3.2-3B model using a single NVIDIA A100 GPU. Following the procedure in [18], we use 6-stage training (stages 0–5) with a batch size of 7. In stage 0, the model is fine-tuned with standard token-level autoregressive training for 2 epochs; in each later stage k ($k = 1, \ldots, 5$), the first k reasoning steps are replaced by k latent embeddings and the model is trained for 1 epoch. **(2) Decoder training.** we distill a LLaMA-3.2-1B decoder from a teacher LLaMA-3.2-3B on a single NVIDIA A100 GPU, using the same 6-stage training (1 epoch per stage) with batch size 7. **(3) Cache inference.** to determine profile similarity during cache retrieval, we employ the pre-trained language model [35]. We test the exploration rate in [0.3, 0.5, 0.7] and select 0.5 based on the efficiency and quality trade-off. We test λ in [0.01, 0.03, 0.05, 0.07] and select 0.05 based on the best performance. The number of search rounds is sampled uniformly between 1 and 3.

4.2.3 ***Baselines.*** We compare our method with LLM-based human mobility simulation baselines including:

- **CoPB [38]** is a mobility simulation framework guides LLMs through reasoning stages to generate mobility activities.
- **Urban-Mobility-LLM (UML) [1]** is a method that synthesizes travel survey data by prompting LLMs to generate individual mobility patterns.
- **LLMob [19]** uses LLM-based agents for personal mobility simulation, enhanced by self-consistency and retrieval strategies.
- **CitySim [3]** is an LLM-driven urban agent simulation framework that generates daily activity through planning, memory, needs, and spatial decision-making.

We provide additional details on the adaptation and implementation of these baselines in Appendix A.1.

We also compare three variants of our model: (1) **w/o LE** removes the latent-space evaluator and selects branches with a plain similarity function instead; (2) **w/o MD** removes the mobility-law distillation loss; and (3) **w/o LD** removes the lightweight decoder and decodes latent embeddings with the original LLM.

4.2.4 ***Metric.*** We evaluate our model from two axes: efficiency and simulation quality.

Efficiency evaluation: We measure efficiency in three ways. (1) *Inference time* (s/trajectory): the average inference time required to generate a single simulated trajectory. For a fair comparison, every method runs sequentially, without parallel or batched execution. (2) *Tokens* (tokens/s): the number of output tokens generated per second during inference. (3) *Monetary cost* ($/trajectory): the average cost per trajectory when generating 20K trajectories. For the API-based baselines (UML, CoPB, LLMob, CitySim), we estimate the average input and output tokens per trajectory and apply GPT-4o-mini's published rates of $0.15 per 1M input tokens and $0.60 per 1M output tokens. For our method, which runs a local LLM, we multiply the average inference time per trajectory by a GPU rate of $0.50/hour on an A6000.

Quality evaluation: To evaluate the similarity between generated and real-world mobility data, we compute the Jensen–Shannon divergence (JSD) between their distributions on five key metrics following existing work [16, 38, 39]: (1) *Radius of gyration* measures the spatial dispersion of individual mobility. We compute the radius of gyration for each individual as the root mean square distance from all visited points to the individual's trajectory centroid. (2) *Stay duration* measures how long individuals remain at each visited location. We compute the duration distribution from the time

Table 1: Overall performance on the Beijing and NYC check-in datasets. Bold indicates the best result, and underline indicates the second-best result. Columns marked with ↓ indicate that lower values are better, while columns with ↑ indicate that higher values are better.

Dataset	Method	Efficiency			Quality				
		Inference time ↓	Tokens ↑	Cost (1e-2) ↓	Radius ↓	Duration ↓	Distance ↓	Locfreq ↓	Odsim ↓
Beijing	CoPB	45.8700±2.7767	40.4406±1.9513	0.6129±0.0045	0.0863±0.0037	0.0314±0.0005	0.0315±0.0004	0.0216±0.0002	0.3213±0.0013
	UML	8.6500±0.2121	71.4088±1.5432	0.0700±0.0017	0.0823±0.0141	**0.0194±0.0004**	0.0569±0.0051	0.0191±0.0016	0.2887±0.0029
	CitySim	35.2400±1.7536	74.5850±1.0536	0.2445±0.0006	0.0746±0.0139	0.0252±0.0004	0.0287±0.0058	0.0328±0.0024	0.3445±0.0151
	LLMob	18.7467±0.7679	54.8446±2.1229	0.2287±0.0022	0.0986±0.0097	0.0322±0.0003	0.0298±0.0055	0.0382±0.0015	0.3606±0.0081
	MobCache	**1.2723±0.0232**	**119.4150±3.3305**	**0.0177±0.0003**	**0.0592±0.0033**	0.0218±0.0002	**0.0271±0.0048**	**0.0189±0.0036**	**0.2634±0.0108**
NYC	CoPB	38.6933±1.4975	46.9200±1.7446	0.6222±0.0007	0.1675±0.0063	N/A	0.0524±0.0058	0.0286±0.0024	0.1784±0.0093
	UML	8.7000±0.1414	70.9510±3.2031	0.0703±0.0023	0.1665±0.0178	N/A	0.0260±0.0057	0.0236±0.0023	0.1517±0.0204
	CitySim	37.8250±1.2233	68.3798±1.4564	0.2661±0.0018	0.1240±0.0281	N/A	0.0685±0.0092	0.0231±0.0029	0.2325±0.0131
	LLMob	15.3300±0.6180	57.5600±2.0219	0.3474±0.0225	0.1331±0.0052	N/A	0.0425±0.0058	0.0855±0.0021	0.2167±0.0115
	MobCache	**1.4259±0.0239**	**108.7361±1.1764**	**0.0198±0.0003**	**0.1082±0.0146**	N/A	**0.0206±0.0016**	**0.0216±0.0003**	**0.1485±0.0168**

intervals between consecutive movements. (3) *Distance* measures the distance between consecutive visited locations. We calculate it as the geographical distance between two consecutive locations in a trajectory. (4) *Location frequency* (Locfreq) measures how well the generated trajectories replicate the spatial distribution of visits. We compute location frequency as the JSD between the spatial visit frequency distributions of the generated and real trajectories. (5) *Origin-destination similarity* (Odsim) measures the similarity between generated and real trajectories in terms of OD travel patterns. It is computed by comparing the normalized frequency distributions of OD pairs using JSD.

4.3 Overall performance

RQ1: *How does MobCache perform compared with existing LLM-based mobility simulation baselines in terms of efficiency and simulation quality?*

4.3.1 ***Performance on Beijing dataset.*** Table 1 reports the comparison between our method and various baselines on the Beijing dataset. **In terms of efficiency**, our method outperforms all the baselines across multiple metrics, achieving at least a 85.29% improvement in inference time, a 60.10% improvement in tokens/s, and a 74.71% reduction in cost. It is worth mentioning that CoPB has a longer inference time mainly because it adopts an explicit multi-stage reasoning process, where each time step sequentially performs intention-confidence ranking, activity selection, and duration estimation while recursively conditioning on previously generated activities. Regarding simulation quality, our method achieves comparable results to LLM-based approaches, demonstrating the effectiveness of our method.

4.3.2 ***Performance on NYC check-in dataset.*** We also conduct experiments on the NYC POI check-in dataset [50]. Specifically, we simulate users' profiles based on U.S. Census demographics [5], and use these profiles as inputs to generate human mobility trajectories. The NYC results use slightly different evaluation metrics because this dataset is based on check-in records, where intervals between check-ins may not accurately reflect actual stay durations. Therefore, duration is not compared on the NYC dataset. The results are shown in Table. 1. In terms of efficiency, our method outperforms all baselines across multiple metrics, achieving at least a 83.61% reduction in inference time, an 53.26% increase in tokens/s, and a 71.83% improvement in cost efficiency. In terms of simulation quality, our method achieves performance comparable to LLM-based approaches, demonstrating the effectiveness of our framework.

4.4 Evaluation of decoding results via LLMs

RQ2: *Can the decoded trajectories preserve profile consistency and mobility behavioral coherence?* Beyond quantitative metrics, we further assess decoded trajectories from two further angles: (1) profile consistency, whether the generated mobility activities align with the user's profile, and (2) mobility behavioral coherence, whether the generated sequence remains temporally, spatially, and semantically consistent after latent-space decoding.

To evaluate profile consistency, we employ an external LLM evaluator [27, 53]. For each sample, the evaluator is provided with the user profile and the decoded mobility activity sequence, and is asked to determine whether the sequence is plausible for that individual. We then compute the proportion of sequences receiving a positive judgment. The results show that 91% of the trajectories generated by our method are considered consistent with the associated user profiles, compared with 82% for MobCache w/o LE. This improvement suggests that the latent-space evaluator effectively guides cache-path selection and helps preserve profile-aware behavioral patterns during decoding.

To evaluate semantic consistency under latent recombination, we add an experiment on behavioral coherence. We compare MobCache with a random-recombination baseline that merges latent embeddings without any compatibility check. An independent LLM evaluator judges whether each generated trajectory is coherent as a whole, temporally, spatially, and semantically. Our method achieves a coherence rate of 87.5%, substantially higher than the 52.0% obtained by random recombination. These results indicate that the proposed latent-space evaluation mechanism preserves meaningful behavioral structure and mitigates the semantic inconsistencies that may arise from unconstrained latent recombination.

4.5 Importance of cache diversity

RQ3: *How important is cache diversity for preserving fidelity and behavioral diversity in mobility simulation?* In Section 2.1, we argue

that cache diversity is essential for realistic mobility simulation. We test this by comparing MobCache with the group-based approach [8] on the Beijing dataset, from two angles: population-level fidelity and population-level behavioral diversity.

First, cache diversity improves the fidelity of simulation. Figure 5 reports the JSD between simulated and real distributions for both methods across the five metrics. Averaged over the five, MobCache improves JSD by 55.27% relative to the group-based method, showing that a diverse cache reproduces real mobility statistics more faithfully.

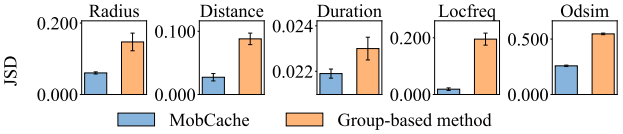

Figure 5: Effect of cache diversity on simulation fidelity.

Second, cache diversity is crucial for preserving behavioral heterogeneity. Figure 6 shows the radius-of-gyration distribution for the real and simulated populations. MobCache tracks the real distribution closely, whereas the group-based method underestimates the highly mobile individuals in the tail and so spans a much narrower range of behaviors. A handful of archetypes cannot capture the full diversity of real movement; by caching and recombining mobility-aware reasoning chains, MobCache generates more varied patterns and better preserves population heterogeneity.

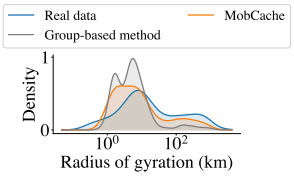

Figure 6: Effect of cache diversity on population diversity.

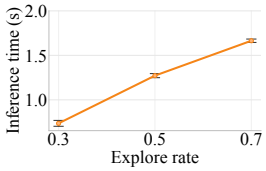

Figure 7: Impact of exploration rate on inference time.

4.6 Cache reuse and exploration analysis

RQ4: *How does MobCache balance cache reuse and exploration to achieve efficient mobility simulation?* To understand where the efficiency gains come from, we analyze the role of cache reuse and exploration in our framework. Unlike conventional caching systems, where efficiency is typically characterized by cache hit and miss rates, our method operates under a retrieval-based reuse mechanism. Specifically, for 10K users outside the cache, every user can consistently find a highly similar cached profile, with similarity scores exceeding 0.977. As a result, strict hit/miss distinctions become less informative, since suitable cache candidates are almost always available.

Instead, the key factor governing efficiency is the exploration rate, which sets how often the system searches for new mobility chains beyond the retrieved ones. We evaluate this effect on the Beijing dataset by varying the exploration rate and measuring both simulation quality and inference time. Figure 7 shows that inference time increases with the exploration rate, as additional mobility chains must be searched and evaluated during simulation. Reducing the exploration rate from 0.7 to 0.3 more than halves the inference time, highlighting exploration as the primary source of online computational overhead. However, lower exploration also reduces simulation quality. As shown in Figure 8, increasing the exploration rate improves simulation quality, but the marginal gains beyond 0.5 are limited relative to the additional computational cost. Therefore, we use an exploration rate of 0.5 in all experiments.

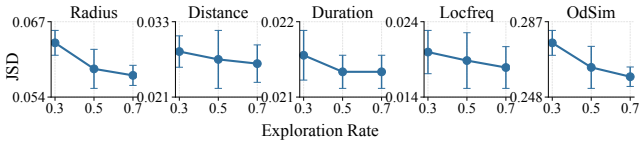

Figure 8: Impact of exploration rate on simulation quality.

4.7 Scalability under large-scale simulation

RQ5: *How well does MobCache scale under larger population sizes and longer simulation horizons?* To further evaluate the scalability of our approach, we conduct two sets of extended experiments on the Beijing dataset by increasing the simulation scale along two dimensions: population size and temporal horizon. In both experiments, the cache is constructed from the original dataset and reused directly without any expansion or retraining, allowing us to assess whether the cached mobility patterns remain effective under substantially larger simulation scales.

Scaling by population size. We first increase the number of users from 1K to 6K, which increases the number of generated trajectories from approximately 20K to 100K. As shown in Figure 9, the performance remains stable despite the substantial increase in simulation scale. For most metrics, the JSD values exhibit only minor fluctuations. Notably, Radius and Odsim show improved alignment with the real data, with their JSD values decreasing from 0.059 to 0.014 and from 0.263 to 0.204, respectively. We attribute these improvements to the larger trajectory collection, which provides more reliable estimates of the underlying mobility distributions and reduces statistical variance. Overall, the results suggest that the cached mobility patterns generalize well to substantially larger populations and that our approach can support large-scale deployment without requiring cache expansion or retraining.

Scaling by temporal horizon. We further evaluate scalability with respect to the simulation duration. Specifically, we extend the temporal horizon from one month to two months, increasing the number of generated trajectories from approximately 20K to 40K while keeping the cache fixed. As shown in Figure 10, the performance remains largely stable under the extended simulation horizon. Across all mobility statistics, the JSD values exhibit only minor variations despite the doubled simulation period. These results demonstrate the robustness of our approach to temporal scaling without requiring cache expansion or retraining.

4.8 Comparison with fine-tuning-based LLM mobility simulation

RQ6: *Can MobCache achieve competitive performance against fine-tuned LLM mobility simulation that uses real-world trajectories?* We compare MobCache with Geo-LLaMA [25], a representative fine-tuning-based LLM mobility simulation that relies on real trajectories for training. This comparison is supplementary to our main

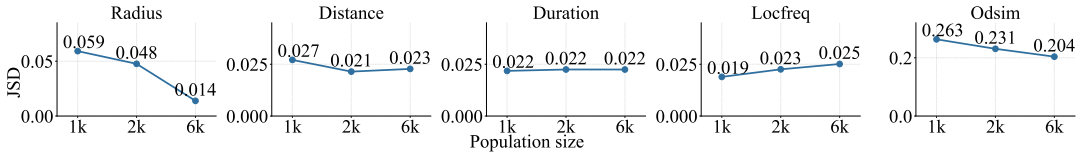

Figure 9: Scaling simulation by population size.

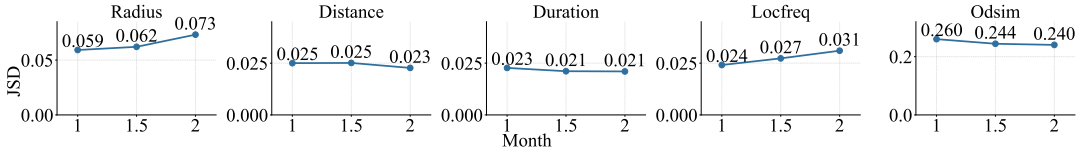

Figure 10: Scaling simulation by temporal horizon.

setting, where MobCache does not use any real trajectories from the target task for training. Following its training-based setting, we fine-tune Geo-LLaMA using 7,000 real-world trajectories from the Beijing dataset. To avoid data leakage, the individuals included in the fine-tuning set are strictly disjoint from those in the test set. Geo-LLaMA inference was performed locally on an NVIDIA RTX A6000 GPU using its LLaMA-2-7B-based implementation.

Figure 11 and Figure 12 shows that MobCache improves both efficiency and simulation quality over Geo-LLaMA. In terms of efficiency, MobCache achieves lower inference time and cost. In terms of simulation quality, MobCache obtains lower JSD across metrics. Although Geo-LLaMA is trained on real trajectories, its training and test users are disjoint; therefore, its performance still depends on how well it generalizes to unseen users. Under this setting, MobCache remains more effective despite not using real trajectories for training.

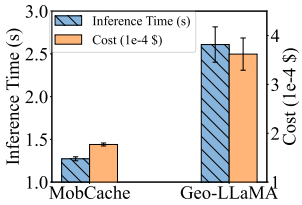

Figure 11: Efficiency comparison (MobCache vs. Geo-LLaMA).

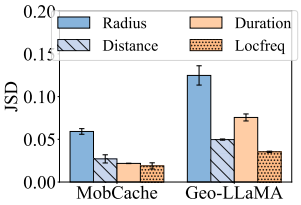

Figure 12: Quality comparison (MobCache vs. Geo-LLaMA).

4.9 Ablation study

RQ7: *How does each key component affect the performance of MobCache?* **Effect of latent-space evaluator:** To investigate the role of the latent-space evaluator, we substitute it with a naive similarity function (i.e., MobCache w/o LE) for determining the validity of connections between latent embeddings. The results (Beijing dataset) in Table 2 show a decline in simulation quality, highlighting the evaluator's critical role in ensuring coherent behavior generation.

Effect of decoder: To evaluate the effectiveness of our lightweight decoder, we use the original LLM in Section 3.3 to decode the reasoning chain (i.e., MobCache w/o LD) on Beijing dataset. As shown in the Table 2, although it achieves better quality on all metrics, it comes at a high computational cost. This shows that our lightweight decoder provides a balance between quality and efficiency. We also evaluate the role of the mobility law distillation (i.e., MobCache w/o MD). The results in Table 2 reveal that removing mobility law constraints leads to a quality performance drop.

4.10 MobCache as a plug-and-play accelerator for existing simulators

RQ8: *Can MobCache be applied to existing mobility simulation to improve efficiency while preserving simulation quality?* MobCache is built to wrap around an existing simulator rather than replace it. As a case study, we apply MobCache to improve the simulation efficiency of Urban-Mobility-LLM (UML) on the Beijing dataset. Specifically, we first use UML to generate initial mobility trajectory data, which is then fed into our framework for LLM fine-tuning (details in Section 3.3). Then we can obtain the required latent embeddings for building the cache. Once the cache is constructed, we can perform efficient mobility simulation.

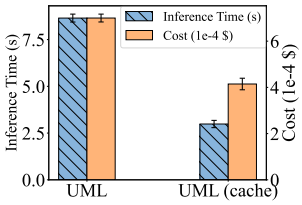

Figure 13: Efficiency comparison (UML vs. UML (cache)).

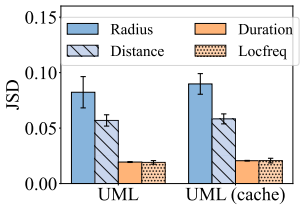

Figure 14: Quality comparison (UML vs. UML (cache)).

We compare UML and its cache-enhanced version (i.e., UML (cache)), accelerated by our MobCache framework. In terms of efficiency, as shown in Figure 13, UML (cache) achieves a simulation speed of 2.985s per trajectory, compared to 8.650s per trajectory for the original UML, resulting in a 65.49 % improvement in simulation speed. In terms of monetary cost, the average expense per trajectory decreases from $\$7.00 \times 10^{-4}$ to $\$4.15 \times 10^{-4}$, resulting in a 40.71% improvement in cost efficiency. In addition, we evaluate quality using four metrics. As shown in Figure 14, the performance of UML (cache) remains comparable to the original UML.

4.11 Cross-city transferability

RQ9: *Can a cache constructed from one city be used for mobility simulation in another city?* We conduct a cross-city transferability experiment to evaluate whether a cache constructed from one city (i.e., Beijing) can effectively accelerate simulations in another city

Table 2: Comparison with Variants of MobCache. Bold scores are for the best values.

Method	Effiency			Quality				
	Inference time ↓	Tokens ↑	Cost (1e-2) ↓	Radius ↓	Duration ↓	Distance ↓	Locfreq ↓	Odsim ↓
w/o LE	1.2710±0.0165	118.8533±2.2022	0.0177±0.0002	0.0619±0.0031	0.0232±0.0002	0.0286±0.0052	0.0195±0.0029	0.2747±0.0097
w/o MD	1.2777±0.0154	119.1867±2.1794	0.0177±0.0002	0.0616±0.0028	0.0226±0.0001	0.0279±0.0055	0.0193±0.0033	0.2738±0.0099
w/o LD	2.1950±0.0477	75.8777±1.0337	0.0309±0.0014	**0.0564±0.0026**	**0.0205±0.0002**	**0.0257±0.0045**	**0.0170±0.0029**	**0.2374±0.0095**
MobCache	**1.2723±0.0232**	**119.4150±3.3305**	**0.0177±0.0003**	0.0592±0.0033	0.0218±0.0002	0.0271±0.0048	0.0189±0.0036	0.2634±0.0108

(i.e., New York). Specifically, we use the cache built from the Beijing dataset in Section 4.3 to accelerate trajectory simulation in New York. We compare MobCache using the NYC cache (i.e., Cache (NYC) for NYC) with MobCache using the Beijing cache (i.e., Cache (BJ) for NYC). For evaluation, we use the NYC POI check-in dataset described in Section 4.3.2 as the test set. It is worth noting that we maintain the same user profile format to enable the retrieval of more similar users from the cache.

From the Figure 15, we observe that, in terms of efficiency, the inference speed on the New York dataset is lower than that on the Beijing dataset. This is because the cache is constructed from Beijing data, resulting in a small portion of New York users failing to find similar matches, which may require additional LLM invocations. In terms of simulation quality, as shown in Figure 16, the performance is slightly lower than that achieved by a cache specifically built for NYC, but remains acceptable. This is because the cache stores the reasoning process rather than city-specific locations, allowing the reasoning and decision-making to be transferred across cities.

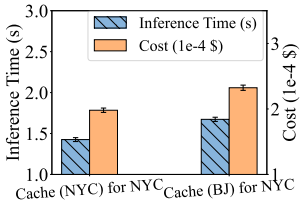

Figure 15: Cross-city: efficiency.

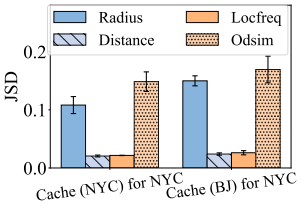

Figure 16: Cross-city: quality.

4.12 Computational cost analysis

RQ10: *How do the one-time training and cache construction costs of MobCache amortize as the simulation scale increases?* We analyze end-to-end computational efficiency on the Beijing dataset, measuring total inference time and total cost across simulation scales. To be fair, we fold in all one-time preparation costs: data initialization, model fine-tuning, and cache construction. Because the cache is reused across simulations, these costs are paid once and amortize as the scale grows. We report the actual cost for the 20K-trajectory setting. For larger scales, we extrapolate from each method's measured per-trajectory cost to estimate city-scale settings that would be too expensive to run in full.

As shown in Figure 17, MobCache exhibits substantially better scalability than competing methods in both runtime and monetary cost. Although cache construction introduces additional fixed overhead at smaller scales, this cost is rapidly amortized as the number of simulated trajectories increases. At the 100K-trajectory scale, MobCache requires only 60.02 hours and $66.46, compared with 240.28 hours and $70.00 for UML, 520.74 hours and $228.70 for LLMob, and 1274.17 hours and $612.90 for CoPB. The efficiency gap further widens as the simulation scale increases, demonstrating the effectiveness of cache reuse in reducing repeated LLM inference.

To put the largest simulation scale into context, according to the 2020 United States Census, Manhattan had a resident population of approximately 1.7 million people [43]. Under the common assumption of generating one trajectory per resident per day, a realistic one-day city-scale simulation would involve roughly 1.7 million trajectories. At this scale, MobCache achieves at least a 6.5× speedup and a 3.4× reduction in monetary cost compared to existing methods. These results demonstrate that our framework become increasingly pronounced at realistic urban scales, making large-scale mobility simulation significantly more practical and cost-effective.

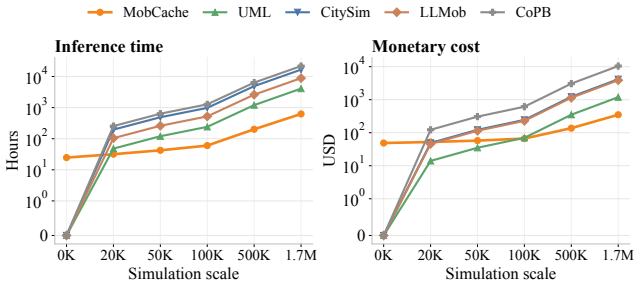

Figure 17: Scalability of total inference time and monetary cost across simulation scales, including one-time training costs.

5 Discussion

Lessons learned. Based on the results from our paper, we summarize the following lessons learned:

- Our reconstructible latent-space cache can augment LLM calls by querying cached results, thereby accelerating large-scale mobility simulation. As shown in Table 1, MobCache achieves comparable performance to state-of-the-art LLM-based methods while significantly reducing inference time and resource consumption.
- Our framework can be generalized to other LLM-based mobility simulation methods. As demonstrated in Figure 13 and Figure 14, MobCache significantly enhances the simulation efficiency of UML while maintaining comparable simulation quality.

Limitation. While our framework effectively accelerates other mobility simulation models, it only applies to models that expose interpretable reasoning steps. In particular, the simulation model must provide accessible step-by-step reasoning, either as text or structured latent representations.

Ethics and privacy. This work focuses on accelerating large-scale mobility simulation rather than individual tracking or prediction. Our approach does not require real trajectory data for supervised model training; instead, real-world data is used only for evaluation.

Moreover, all trajectory data used for evaluation is fully anonymized and does not contain identifiable user information. Our framework relies on user profiles from census statistics or publicly available data sources. All information is highly anonymized and represented at a coarse-grained level, preventing the identification of specific individuals and preserving user privacy.

6 Related work

6.1 Human mobility simulation.

6.1.1 ***Deep learning based human mobility simulation.*** Deep learning-based approaches generate mobility trajectories by learning patterns from historical data. Existing studies mainly follow two directions: sequential prediction, which uses RNNs, LSTMs, and Transformers to model spatio-temporal dependencies [12, 30, 47, 49], and generative simulation, which uses GANs, VAEs, and diffusion models to synthesize trajectories from learned distributions [13, 17, 55]. These methods are efficient for large-scale generation once trained. However, their performance heavily depends on access to large-scale historical mobility datasets. In data-scarce settings, these models often struggle to generalize across new cities, populations, and behavioral contexts. Therefore, we do not directly compare with these methods, as they assume access to large-scale real trajectory data for training, whereas such data are not available in our setting. Nevertheless, we provide an experimental comparison in Section 4.8 to illustrate the relative performance of our approach against these methods.

6.1.2 ***LLM-based human mobility simulation.*** LLM-based human mobility simulation. Existing works explored using LLMs for human mobility simulation by leveraging their knowledge and human-like reasoning capabilities. The advantage of these approaches is that they do not require large-scale real-world mobility data for training. These works [1, 10, 19, 21, 25, 29, 31, 34, 38] guide LLMs to simulate human-like mobility intention reasoning step by step and then produce realistic mobility activity sequences. For example, CoPB [38] is an intention and planing based framework that enables LLMs to generate human mobility trajectories through step-by-step reasoning. Although these works produce realistic outputs, the reliance on LLMs makes the simulation expensive. This work [19] design a LLM-based agent framework for personal mobility generation, combining self-consistency and retrieval strategies to align language models with real-world human activity for accurate and interpretable urban mobility simulation.

To reduce the cost of LLM-based simulation, some works [8] design a group-based approach, where people are clustered into coarse-grained groups based on profiles and the LLM is invoked once per group to generate shared mobility activities. However, this approach limits diversity, since people in the same group share identical behaviors. Other works focus on optimizing the efficiency of API interactions [34, 48]. For example, OpenCity [48] accelerates LLM-based simulation by combining I/O multiplexing and TCP connection pooling to parallelize LLM requests. AgentSociety [34] accelerates GPT API calls by using Agent Grouping, Ray with asyncio for asynchronous distributed execution, MQTT-based message reuse, and a unified interface for local and remote models, enabling large-scale agent simulations on commodity hardware. However, the requirement of querying the LLM API for each agent at every simulation step remains, resulting in high cumulative cost.

6.2 Caches for LLMs.

When referring to "cache" in the context of LLMs, the most common form is key-value (KV) caching. KV cache stores the intermediate attention states of previously processed tokens, allowing faster autoregressive decoding without re-computing hidden states. Techniques such as prefix caching [22, 26, 28] reuse KV pairs for initial prompt tokens, while full KV [15] reuse extends this idea to non-prefix positions via positional embedding shifts. Our method is orthogonal to traditional KV caching. While KV cache accelerates decoding at the token level within a fixed prompt, our latent cache operates at a higher level by storing latent reasoning steps. This enables flexible reuse across different simulation queries. Importantly, the two approaches are complementary. Our method can potentially benefit further by incorporating KV cache for additional decoding speedup. In addition, several existing methods leverage query similarity to cache or retrieve useful information, thereby improving efficiency or simulation quality in LLMs [2, 24, 33]. For example, MemGPT [33] introduces a memory system that simulates long-term memory, retrieving context dynamically during extended interactions. Unlike retrieval methods based on text similarity, our method enables latent-space reasoning reuse, supporting efficient and compositional simulation.

7 Conclusion

We presented MobCache, a caching framework that makes large-scale LLM-based mobility simulation practical. Its central idea is to cache the *reasoning* behind mobility behavior rather than the generated trajectories, and to do so in latent space rather than in language. Caching reasoning lets a small set of cached chains be recombined into many distinct trajectories, which preserves population diversity; keeping that reasoning in latent space lets us enforce the spatial and temporal constraints that are easily broken when reasoning steps are recombined as text. MobCache realizes this idea with a reconstructible cache of latent reasoning embeddings, searched as a tree for flexible reuse, and a lightweight decoder, distilled under mobility-law constraints, that turns a reconstructed chain back into a trajectory without repeated calls to the original LLM. Across two datasets, MobCache matches or exceeds prior LLM-based simulators on trajectory quality while running several times faster and cheaper, and the gap widens at city scale. Because the cache stores reasoning rather than city-specific locations, it transfers to a new city and scales to larger populations and longer horizons with no retraining, and it can be dropped into an existing simulator to accelerate it without changing the host method.

References

[1] Prabin Bhandari, Antonios Anastasopoulos, and Dieter Pfoser. 2024. Urban mobility assessment using llms. In *Proceedings of the 32nd ACM International Conference on Advances in Geographic Information Systems*. 67–79.

[2] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*. PMLR, 2206–2240.

[3] Nicolas Bougie and Narimawa Watanabe. 2025. Citysim: Modeling urban behaviors and city dynamics with large-scale llm-driven agent simulation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*. 215–229.

[4] Stacey Bricka, Timothy Reuscher, Paul Schroeder, Mitchell Fisher, Justina Beard, and Xiaoyuan Layla Sun. 2024. Summary of travel trends: 2022 national household travel survey. (2024).

[5] U.S. Census Bureau. 2025. American Community Survey (ACS). https://www.census.gov/programs-surveys/acs.html.

[6] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176* (2023).

[7] ChicagoGov. 2023. Scooter Sharing in Chicago. https://www.chicago.gov/city/en/depts/cdot/supp_info/escooter-share-pilot-project.html

[8] Ayush Chopra, Shashank Kumar, Nurullah Giray-Kuru, Ramesh Raskar, and Arnau Quera-Bofarull. 2024. On the limits of agency in agent-based models. *arXiv preprint arXiv:2409.10568* (2024).

[9] Camille Couturier, Spyros Mastorakis, Haiying Shen, Saravan Rajmohan, and Victor Rühle. 2025. Semantic Caching of Contextual Summaries for Efficient Question-Answering with Language Models. *arXiv preprint arXiv:2505.11271* (2025).

[10] Yuwei Du, Jie Feng, Jian Yuan, and Yong Li. 2025. CAMS: A CityGPT-Powered Agentic Framework for Urban Human Mobility Simulation. *arXiv preprint arXiv:2506.13599* (2025).

[11] Zipei Fan, Xuan Song, Yinghao Liu, Zhiwen Zhang, Chuang Yang, Quanjun Chen, Renhe Jiang, and Ryosuke Shibasaki. 2020. Human mobility based individual-level epidemic simulation platform. *SIGSPATIAL Special* 12, 1 (2020), 34–40.

[12] Jie Feng, Yong Li, Chao Zhang, Funing Sun, Fanchao Meng, Ang Guo, and Depeng Jin. 2018. Deepmove: Predicting human mobility with attentional recurrent networks. In *Proceedings of the 2018 world wide web conference*. 1459–1468.

[13] Jie Feng, Zeyu Yang, Fengli Xu, Haisu Yu, Mudan Wang, and Yong Li. 2020. Learning to simulate human mobility. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 3426–3433.

[14] Haoyu Geng, Guanjie Zheng, Zhengqing Han, Hua Wei, and Zhenhui Li. 2022. HMES: A Scalable Human Mobility and Epidemic Simulation System with Fast Intervention Modeling. In *2022 IEEE Smartworld, Ubiquitous Intelligence & Computing, Scalable Computing & Communications, Digital Twin, Privacy Computing, Metaverse, Autonomous & Trusted Vehicles (SmartWorld/UIC/ScalCom/DigitalTwin/PriComp/Meta)*. IEEE, 468–475.

[15] In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. 2024. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems* 6 (2024), 325–338.

[16] Marta C Gonzalez, Cesar A Hidalgo, and Albert-Laszlo Barabasi. 2008. Understanding individual human mobility patterns. *nature* 453, 7196 (2008), 779–782.

[17] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. 2018. Social gan: Socially acceptable trajectories with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2255–2264.

[18] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769* (2024).

[19] WANG JIAWEI, Renhe Jiang, Chuang Yang, Zengqing Wu, Ryosuke Shibasaki, Noboru Koshizuka, Chuan Xiao, et al. 2024. Large language models as urban residents: An llm agent framework for personal mobility generation. *Advances in Neural Information Processing Systems* 37 (2024), 124547–124574.

[20] Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. 2024. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *arXiv preprint arXiv:2404.12457* (2024).

[21] Chenlu Ju, Jiaxin Liu, Shobhit Sinha, Hao Xue, and Flora Salim. 2025. Trajllm: A modular llm-enhanced agent-based framework for realistic human trajectory simulation. In *Companion Proceedings of the ACM on Web Conference 2025*. 2847–2850.

[22] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*. 611–626.

[23] Pierre-Yves Lajoie, Bobak Hamed Baghi, Sachini Herath, Francois Hogan, Xue Liu, and Gregory Dudek. 2024. PEOPLEx: Pedestrian opportunistic positioning leveraging IMU, UWB, BLE and WiFi. In *ICC 2024-IEEE International Conference on Communications*. IEEE, 3518–3523.

[24] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[25] Siyu Li, Toan Tran, Haowen Lin, John Krumm, Cyrus Shahabi, Lingyi Zhao, Khurram Shafique, and Li Xiong. 2024. Geo-llama: Leveraging llms for human mobility trajectory generation with spatiotemporal constraints. *arXiv preprint arXiv:2408.13918* (2024).

[26] Shu Liu, Asim Biswal, Audrey Cheng, Xiangxi Mo, Shiyi Cao, Joseph E Gonzalez, Ion Stoica, and Matei Zaharia. 2024. Optimizing llm queries in relational workloads. *CoRR* (2024).

[27] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-eval: NLG evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 conference on empirical methods in natural language processing*. 2511–2522.

[28] Yuhan Liu, Hanchen Li, Kuntai Du, Jiayi Yao, Yihua Cheng, Yuyang Huang, Shan Lu, Michael Maire, Henry Hoffmann, Ari Holtzman, et al. 2023. Cachegen: Fast context loading for language model applications. *CoRR* (2023).

[29] Yifan Liu, Xishun Liao, Haoxuan Ma, Brian Yueshuai He, Chris Stanford, and Jiaqi Ma. 2024. Human Mobility Modeling with Household Coordination Activities under Limited Information via Retrieval-Augmented LLMs. *arXiv preprint arXiv:2409.17495* (2024).

[30] Yingtao Luo, Qiang Liu, and Zhaocheng Liu. 2021. Stan: Spatio-temporal attention network for next location recommendation. In *Proceedings of the web conference 2021*. 2177–2185.

[31] Xinyi Mou, Xuanwen Ding, Qi He, Liang Wang, Jingcong Liang, Xinnong Zhang, Libo Sun, Jiayu Lin, Jie Zhou, Xuanjing Huang, et al. 2024. From individual to society: A survey on social simulation driven by large language model-based agents. *arXiv preprint arXiv:2412.03563* (2024).

[32] Farshid Nooshi and Suining He. 2025. Multi-Agent Reinforcement Learning for Dynamic Mobility Resource Allocation with Hierarchical Adaptive Grouping. *arXiv preprint arXiv:2507.20377* (2025).

[33] Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. 2023. MemGPT: Towards LLMs as Operating Systems. (2023).

[34] Jinghua Piao, Yuwei Yan, Jun Zhang, Nian Li, Junbo Yan, Xiaochong Lan, Zhihong Lu, Zhiheng Zheng, Jing Yi Wang, Di Zhou, et al. 2025. Agentsociety: Large-scale simulation of llm-driven generative agents advances understanding of human behaviors and society. *arXiv preprint arXiv:2502.08691* (2025).

[35] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 3982–3992. doi:10.18653/v1/D19-1410

[36] Yangjun Ruan, Neil Band, Chris J Maddison, and Tatsunori Hashimoto. 2025. Reasoning to learn from latent thoughts. *arXiv preprint arXiv:2503.18866* (2025).

[37] Abhishek Shah. 2025. Navigating the LLM Cost Maze: A Q2 2025 Pricing and Limits Analysis. https://ashah007.medium.com/navigating-the-llm-cost-maze-a-q2-2025-pricing-and-limits-analysis-80e9c832ef39. Accessed: 2025-07-31.

[38] Chenyang Shao, Fengli Xu, Bingbing Fan, Jingtao Ding, Yuan Yuan, Meng Wang, and Yong Li. 2024. Chain-of-planned-behaviour workflow elicits few-shot mobility generation in llms. *arXiv preprint arXiv:2402.09836* (2024).

[39] Chaoming Song, Tal Koren, Pu Wang, and Albert-László Barabási. 2010. Modelling the scaling properties of human mobility. *Nature physics* 6, 10 (2010), 818–823.

[40] Heng Tan, Yukun Yuan, Shuxin Zhong, and Yu Yang. 2023. Joint rebalancing and charging for shared electric micromobility vehicles with energy-informed demand. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 2392–2401.

[41] Wenhui Tan, Jiaze Li, Jianzhong Ju, Zhenbo Luo, Jian Luan, and Ruihua Song. 2025. Think Silently, Think Fast: Dynamic Latent Compression of LLM Reasoning Chains. *arXiv preprint arXiv:2505.16552* (2025).

[42] Eran Toch, Boaz Lerner, Eyal Ben-Zion, and Irad Ben-Gal. 2019. Analyzing large-scale human mobility data: a survey of machine learning methods and applications. *Knowledge and Information Systems* 58, 3 (2019), 501–523.

[43] U.S. Census Bureau. n.d.. QuickFacts: New York County, New York; New York city, New York. https://www.census.gov/quickfacts/fact/table/newyorkcountynewyork,newyorkcitynewyork/PST045225

[44] Yu Wang, Tongya Zheng, Shunyu Liu, Zunlei Feng, Kaixuan Chen, Yunzhi Hao, and Mingli Song. 2024. Spatiotemporal-augmented graph neural networks for human mobility simulation. *IEEE Transactions on Knowledge and Data Engineering* 36, 11 (2024), 7074–7086.

[45] Wayne Wu, Honglin He, Jack He, Yiran Wang, Chenda Duan, Zhizheng Liu, Quanyi Li, and Bolei Zhou. 2024. Metaurban: An embodied ai simulation platform for urban micromobility. *arXiv preprint arXiv:2407.08725* (2024).

[46] Wayne Wu, Honglin He, Chaoyuan Zhang, Jack He, Seth Z Zhao, Ran Gong, Quanyi Li, and Bolei Zhou. 2025. Towards autonomous micromobility through scalable urban simulation. In *Proceedings of the Computer Vision and Pattern*

Recognition Conference. 27553–27563.

[47] Yuan Xu, Jiajie Xu, Jing Zhao, Kai Zheng, An Liu, Lei Zhao, and Xiaofang Zhou. 2022. Metaptp: An adaptive meta-optimized model for personalized spatial trajectory prediction. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2151–2159.

[48] Yuwei Yan, Qingbin Zeng, Zhiheng Zheng, Jingzhe Yuan, Jie Feng, Jun Zhang, Fengli Xu, and Yong Li. 2024. Opencity: A scalable platform to simulate urban activities with massive llm agents. *arXiv preprint arXiv:2410.21286* (2024).

[49] Dingqi Yang, Benjamin Fankhauser, Paolo Rosso, and Philippe Cudre-Mauroux. 2020. Location prediction over sparse user mobility traces using rnns. In *Proceedings of the twenty-ninth international joint conference on artificial intelligence*. 2184–2190.

[50] Dingqi Yang, Daqing Zhang, Vincent W Zheng, and Zhiyong Yu. 2014. Modeling user activity preference by leveraging user spatial temporal characteristics in LBSNs. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45, 1 (2014), 129–142.

[51] Zhaofan Zhang, Yanan Xiao, Lu Jiang, Dingqi Yang, Minghao Yin, and Pengyang Wang. 2024. Spatial-temporal interplay in human mobility: A hierarchical reinforcement learning approach with hypergraph representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 9396–9404.

[52] Gang Zhao, Ximing Zhang, Chenji Lu, Hui Zhao, Tianshu Wu, Pengjie Wang, Jian Xu, and Bo Zheng. 2025. Explainable LLM-driven Multi-dimensional Distillation for E-Commerce Relevance Learning. In *Companion Proceedings of the ACM on Web Conference 2025*. 631–640.

[53] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems* 36 (2023), 46595–46623.

[54] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody_Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. 2023. Efficiently Programming Large Language Models using SGLang. (2023).

[55] Yuanshao Zhu, Yongchao Ye, Shiyao Zhang, Xiangyu Zhao, and James Yu. 2023. Difftraj: Generating gps trajectory with diffusion probabilistic model. *Advances in Neural Information Processing Systems* 36 (2023), 65168–65188.

A Appendix

A.1 Baseline adaptation details

We provide additional details on how each baseline method is adapted to our data setting to ensure a fair comparison. To further ensure comparability, all baseline methods use GPT-4o-mini as the underlying LLM.

CoPB: We replaced their fine-tuned LLM with an API-based LLM model for fair comparison with other methods, while preserving CoPB's step-by-step intention reasoning framework. We also reformulated the input profiles and daily records according to the characteristics of the Beijing and NYC datasets.

CitySim: We adapt CitySim to Beijing and NYC datasets by grounding each simulated person in their demographic profile, home location, and observed visit history. The model then generates daily activity trajectories across the user's observed dates, carrying memory and reflections from previous days to make later simulations more consistent with that person's routines.

UML: We implement the framework by retaining the 19 activity categories defined in the original paper and using a gravity model to assign concrete locations to each inferred activity within our study area.

LLMob: We adapt LLMob to our data setting by replacing real historical trajectories with a self-growing generated memory: each simulated day is stored and retrieved as behavioral context for subsequent days, avoiding direct use of ground-truth mobility history during generation. Phase 2 is further extended to use this memory for motivation inference and trajectory planning, with explicit plan-level and activity-level rationales while preserving the original pattern-motivation-plan framework.

A.2 Prompt example

Daily mobility activity generation prompt example

Profile

`Profile:{profile}`; `Date:{today_date}`; `Nearby Home POIs:{home_poi}`; `Nearby Home Work:{work_poi}`;

Task: Generate the person's mobility activities for the full day based on profile and home and workplace. Each activity must involve physical movement and staying at a new location.

Requirements

- Each activity = one meaningful mobility event: movement to a place + stay + purpose.
- Prefer activities with spatial diversity: different types of locations and distances (short, medium, long).
- Choose realistic activities based on POIs (e.g., gym near home, restaurant near work, hospital visit, etc).
- Encourage inclusion of diverse behavior types: health (e.g., walk, gym), errands, social, entertainment, unusual events.
- The person's day must have between 2 and 9 total mobility activities.
- Output two sections: `Reasoning` and `Final Activities`.

Output Format

`Reasoning:`

- At 12:30 a.m., After late night out, he may go home for sleep. Distance: 8km.
- ...

`Final Activities:` 1. At 12:30 a.m., Return home, 8km...

A.3 Notation

For clarity, Table 3 summarizes the main notations used throughout the paper, including the latent-space reasoning process, reconstructible cache, latent-space evaluator, and lightweight decoder.

Table 3: Summary of notations.

Notation	Description
q	Input prompt containing user profile, date, POIs, and task-specific information.
y	Final mobility activity output.
$Y = (y_1, \ldots, y_L)$	Generated mobility activity sequence.
L	Length of the activity sequence.
r_t	Latent-space reasoning embedding at reasoning step t.
$\hat{r}_t$	Next latent reasoning embedding generated by the latent reasoning model during evaluator training.
$R_{1:t}$	Latent reasoning chain $[r_1, r_2, \ldots, r_t]$.
$f_\theta(\cdot)$	Fine-tuned latent reasoning model.
$g_\xi(\cdot)$	Latent-space evaluator.
σ_t	Evaluator score assigned to candidate reasoning step r_t.
b_t	Supervision label used for evaluator training.
$\mathrm{sim}(\cdot,\cdot)$	Similarity function between latent reasoning embeddings.
$e_\mu(\cdot)$	MLP projector that maps teacher latent embeddings into the lightweight decoder space.
$\hat{R}$	Projected latent reasoning chain used as decoder input.
$P_\rho(\cdot)$	Output distribution of the lightweight decoder.
h_{light}	Hidden representation of the lightweight decoder.
$z_\tau(\cdot)$	Mobility-law prediction network.
$d(\cdot)$	Function extracting mobility-law statistics from activity sequences.
$p_{\text{teacher}}(d(Y))$	Mobility-law distribution derived from activities generated by the teacher decoder.
L_{distill}	Distillation loss for lightweight decoder training.
L_{law}	Mobility-law constraint loss.
L_{total}	Overall training objective.
λ	Weight balancing the distillation loss and mobility-law constraint loss.