# A Deep Learning Approach for Innovating MATSim Plans

Fred Shone - UCL BIG - frederick.shone.17@ucl.ac.uk

Dr Tim Hillel - UCL BIG - tim.hillel@ucl.ac.uk

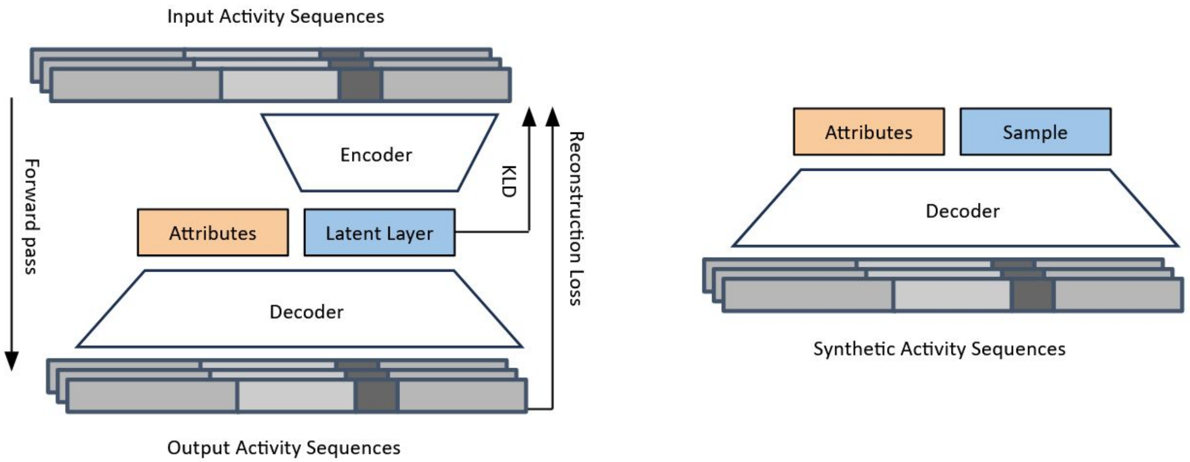
behaviour & infrastructure group

Engineering and Physical Sciences Research Council

# Contents

1. Context!
2. Problem Statement
3. Experiment A - Plan to Score Model
4. Experiment B - Plan to Plan Model

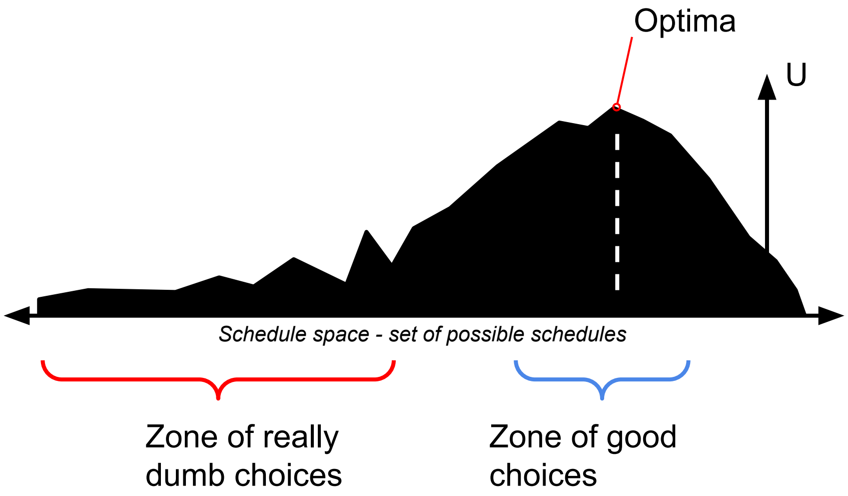
# Context - DGMs for Activity Sequence Generation



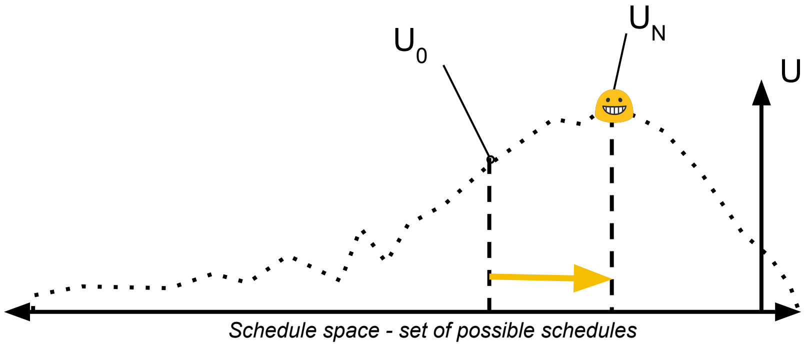

- Fast (on a GPU)
- “Simultaneous” choice generation
- Able to learn realism implicitly from data
- No need to specify structure or alternative specific variables

# Problem Statement

Consider all the possible utility scores an agent can experience in simulation, depending on their mode, route and time choices *(but ignoring other agents for simplicity)*:

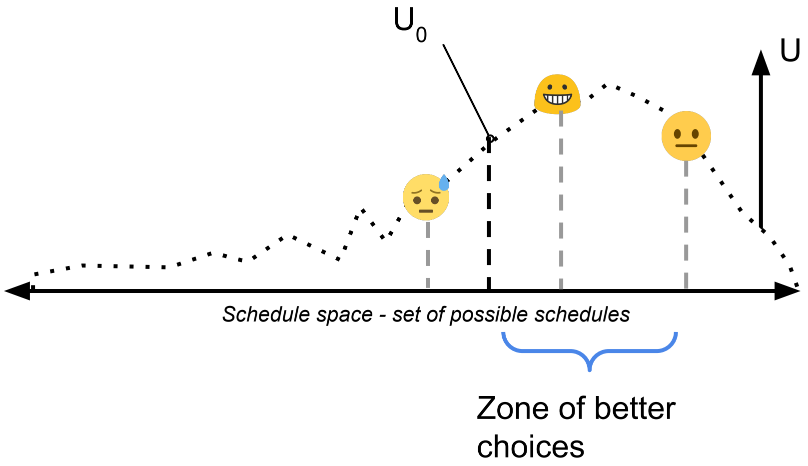



At iteration 0 we start at some initial plan with $U_0$. At some iterations an *innovation* or *innovations* take place, with which we hope to discover a better plan:



# Problem Statement

But what are the odds that our innovation strategies are proposing good plans?




Note:

Agents do not learn from bad experiences, so every simulated bad plan is a wasted iteration.

Bad plans also “poison/taint” the innovations made by other agents. Which is why we limit the overall rate of innovation/exploration.

# Quick Experiment

Existing MATSim model of Sheffield, UK. 120,000 agents. Active modes, routed cycling, transit inter-modal access-egress.

We consider the best utility of agents after intervals of 50 iterations. After each interval we report the probability that an agent finds a better plan. We can think of this as the *efficiency* of the strategies responsible for innovations.

| **Iterations** | **% agents find a better plan** | **…per innovation*** | **…per iteration** |
|---|---|---|---|
| 50 -> 100 | 73% | 4.9% | 1.5% |
| 100 -> 150 | 62% | 4.1% | 1.2% |
| 150 -> 200** | 56% | 3.7% | 1.1%*** |

* Innovation rate (probability of mode, route or time innovation) is 30%.

** Final iterations include cool down/annealing.

*** Arguably this should be zero for final iterations as agents arrive at their optima and cannot do better.

# Innovation Strategies

Consider the approaches to plan innovations, ie the *"strategies"*:

| Strategy Name | Type | Critique |
|---|---|---|
| **ChangeExpBeta** | Exploit | This is required to (i) update plan scores for new choices by other agents and (ii) create a realistic simulation. |
| **RouteChoice** | Shortest path | Edge costs based on previous iteration/s. Edge costs need to be complete/correct for each agent. Greedy. |
| **ModeChoice** | Random | Does not use previous experience by agent or other agents. No/limited consideration of relationship of other choices within plan. |
| **TimeChoice** | Random walk | |
| ***DMC extension*** | *Exploit* | *Mode innovations only. Requires calculation/estimation of alternatives/utilities. No consideration of relationship of other choices types within plan.* |

# Implications

- Inefficient strategies for innovations = slow discovery of *more* optimal plans.
- Innovation rate has to be limited, else supply simulation is misleading.
- Time innovation is often removed/restricted.
- Hardcoded rules to reduce bad mode choices.
- Need high quality initial plans.
- Sim size limited.
- Can’t reasonably add more complex choices to big sims.

# The Problem Statement

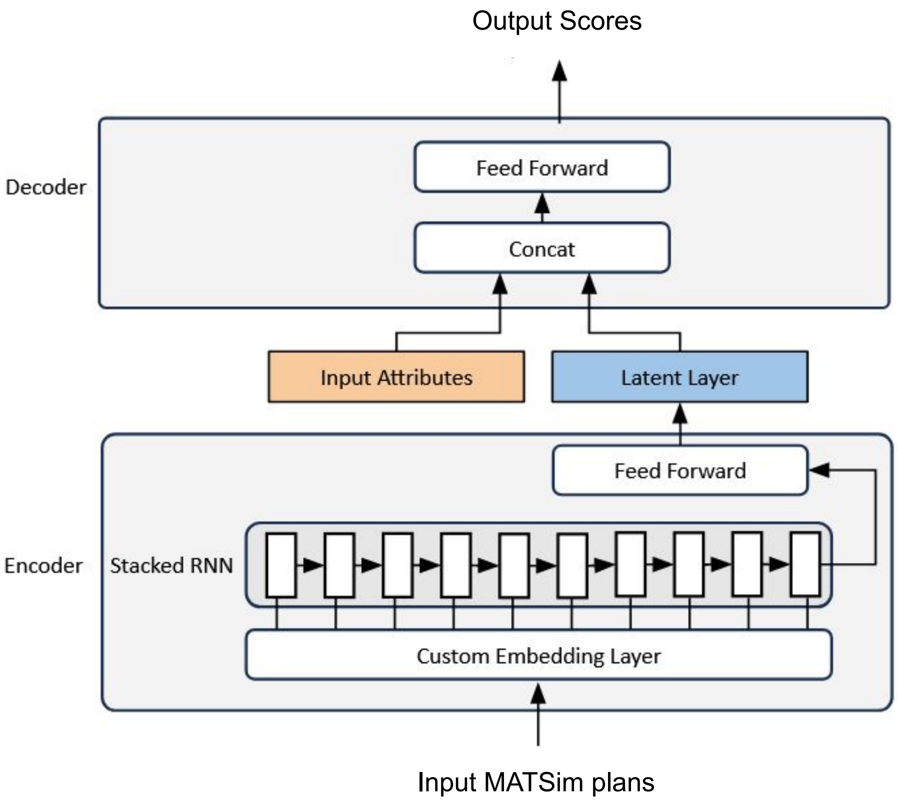
**Can we approximate the distribution of agent utilities over their schedule spaces such that innovations can generate schedules that are more likely to have a higher utility?"**

Can think of this as wanting to improve the "*efficiency*" of innovations.

Find new approach for **mode** and **time** innovations that:

- Considers combined choice**(s)** within a plan - *"simultaneous"*
- Considers experience by other agents - *"shared"*
- Considers experience from previous iterations - *"remembered"*

# Experiment A - Plan to Score Model




Plans are encoded as sequences of activities and trips, padded with special tokens to make the lengths consistent. Each component has a duration, mode and distance.

Attributes are one-hot encoded.

The encoder block uses a learnt embedding for the activity and mode types. Followed by stacked LSTM layers.

This intermediate "latent" layer is concatenated with the encoded attributes, then fed through a regression block of fully connected layers to output a utility score for each sequence. Mean squared error is used for loss.

Many more details at github/fredshone/caveat

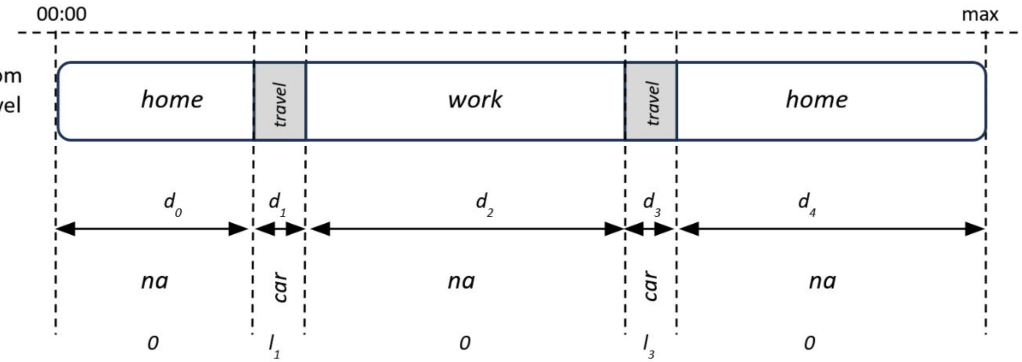
# Encoding MATSim Plans

(i) 24-hour long activity sequences are assembled from trip information. Composed of activity types and travel (categorical):

(ii) Extract durations, normalise with max duration:

(iii) Extract modes (categorical):

(iv) Extract distances, normalise with max distance:




(v) Combine into vectors:

| | | | | |
|---|---|---|---|---|
| $a_0,d_0,m_0,l_0$ | $a_1,d_1,m_1,l_1$ | $a_2,d_2,m_2,l_2$ | $a_3,d_3,m_3,l_3$ | $a_4,d_4,m_4,l_4$ |

(vi) Sequence padded, up to some maximum length, with special start and end of sequence characters:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| <S> | $a_0,d_0,m_0,l_0$ | $a_1,d_1,m_1,l_1$ | $a_2,d_2,m_2,l_2$ | $a_3,d_3,m_3,l_3$ | $a_4,d_4,m_4,l_4$ | <E> | <E> | <E> | <E> |

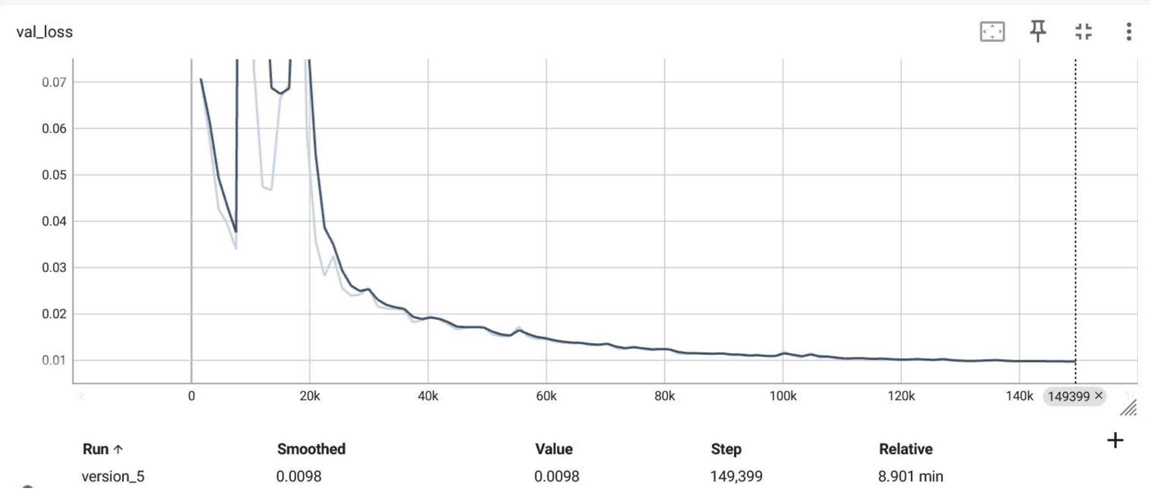
(vii) Final vector representation:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 3 | 4 | 3 | 2 | 1 | 1 | 1 | 1 |
| 0 | .2 | .1 | .3 | .1 | .3 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | .3 | 0 | .3 | 0 | 0 | 0 | 0 | 0 |

github/fredshone/caveat

# Training

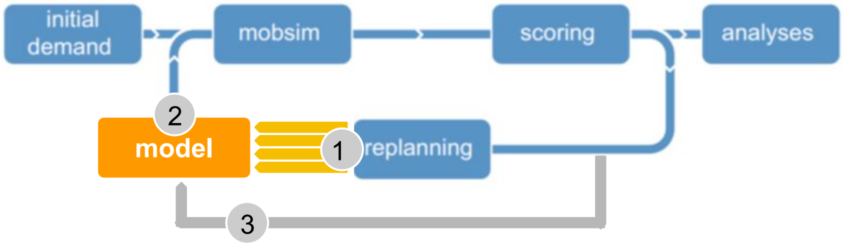
- Extract plans and associated scores from iterations *50, 100, 150 and 200* of an existing MATSim run of 120,000 agents, ~0.5 million plans.
- Train, validate, test split of (80/10/10).
- Training loss (MSE) is **0.0084** (abs error is ~0.1 Utils, equivalent to 0.1GBP).
- Model is learning to approximate **both** the simulation and scoring function.




github/fredshone/caveat

# Example Application/Discussion

1. Replanning module proposes N candidates
2. Model picks best/ignores worst
3. Lagged update model training

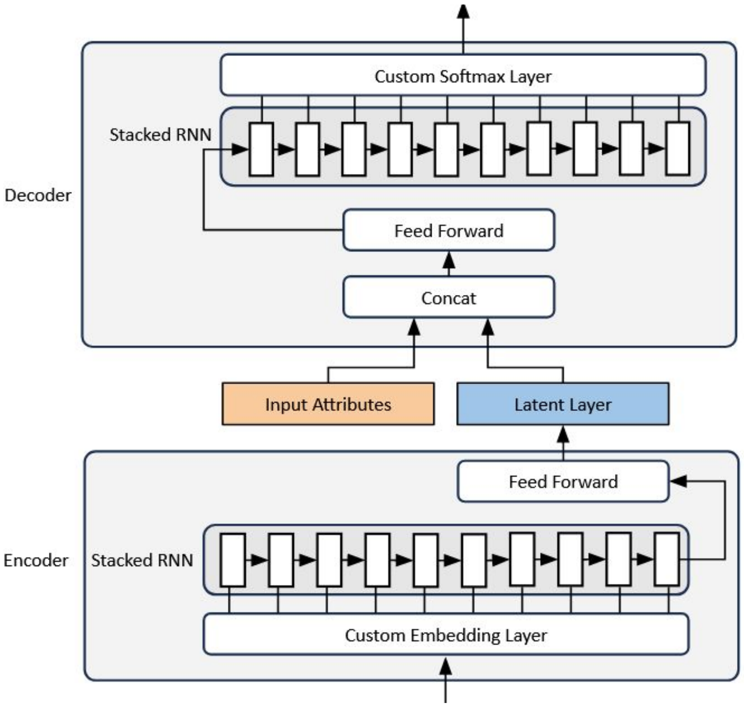



- **Improve quality of plans**
- Most additional compute is on GPU
- **Potential to transfer learn from previous runs**
- Easy to parameterise exploration/exploitation

- Memory implications for creating proposals
- Implementation pretty tough
- **Cold startup/training update**
- **Far simpler models might also work well**

# Experiment B - Plan to Plan Model

Output MATSim plans




Input MATSim plans

Plans are encoded as sequences of activities and trips, padded with special tokens to make the lengths consistent. Each component has a type, duration, mode and distance.

Attributes are one-hot encoded.

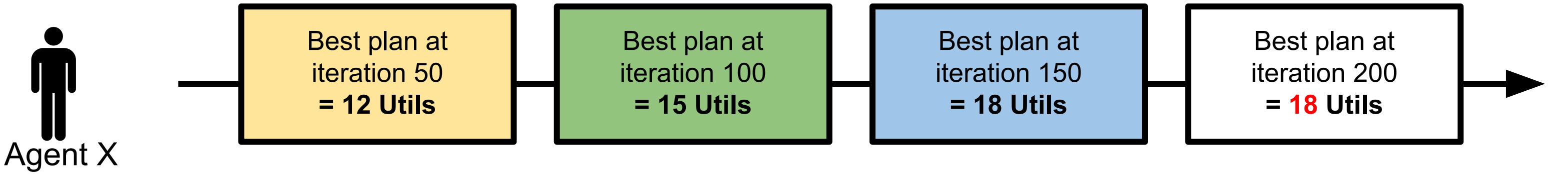
The encoder block uses a learnt embedding for the activity and mode types. Followed by stacked LSTM layers.

This intermediate “latent” layer is concatenated with the encoded attributes, then fed through a decoder block which is similar to the encoder. We use a custom combination of losses.

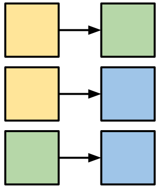
Many more details at github/fredshone/caveat

# Training

- Extract plans and associated scores from iterations *50, 100, 150 and 200* of an existing MATSim run of 120,000 agents.
- Pair together plans for each agent, such that the LHS (input) plan has a lower utility than the RHS (target) plan. In total we create 375,000 pairs.




Training pairs:



# Training

- We train the model to directly infer a *"better plan"* for given input plan.
- Where "better" is a schedule with trip times and modes that will simulate with higher utility.
- We use a 80/10/10 split.
- The test set has an average plan improvement of **4.3** utils.
- We evaluate the utility of generated plans using the matsim scoring function, but note this is an estimate only, as plans have not been simulated.
- The model is able to generate plans with an average improvement of 2.2 utils.
- But on closer inspection, generates an improvement only 36% of the time.
- More pragmatic evaluation:

  **B-** Correctly identifies the target mode 35% of the time.

  **F** Correctly identifies that an activity should be shorter/longer 50% of the time.

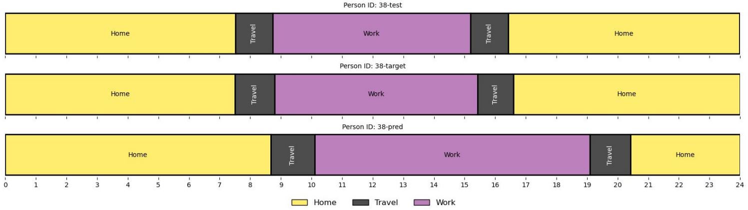
  ~5% failure rate (inferring a schedule with altered activity sequence).

# The Good

Test: 188 utils

Target: 189 utils

Prediction: 197 utils

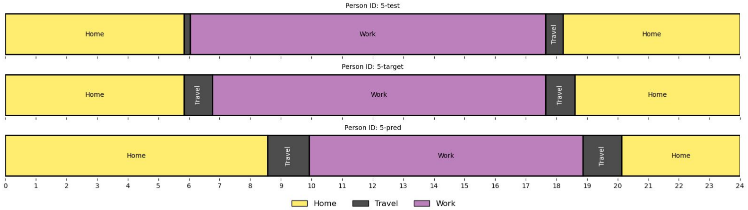



# The Bad

Test: 210 utils

Target: 205 utils

Prediction: 128 utils

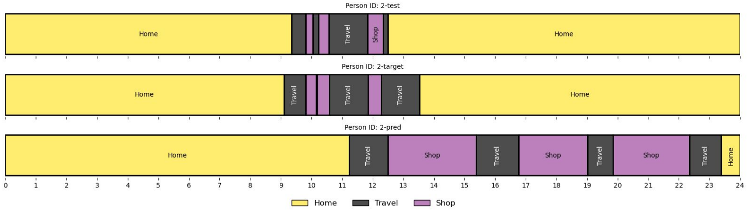



# The Ugly

Test: 163 utils

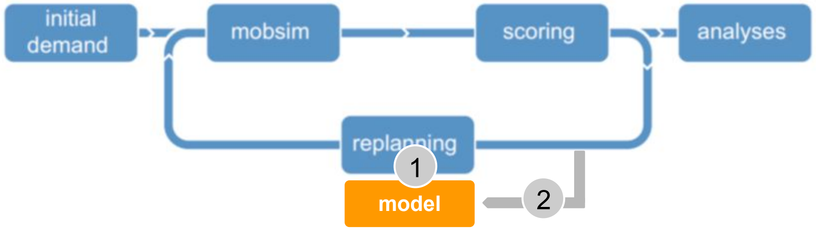
Target: 156 utils

Prediction: 113 utils



# Application/Discussion

1. Use model as an innovation strategy directly
2. Lagged update model training




- Improve quality of plans
- **Additional work on GPU**
- Potential to transfer learning from previous scenarios
- **Can still parameterise exploration/exploitation by combining with other strategies**

- Implementation pretty tough
- **Cannot guarantee viable plans**
- **Potential for failure to explore all choices**
- Cold startup/training update

# Conclusions

- Demonstrated two approaches to improve the "quality" or "efficiency" of innovations for plan modes and times:
  - *Simultaneous* consideration of choices
  - *Shared* experience of choices
  - *Remembered* experience of choices
- Potentially (very?) fast, but some outstanding questions around implementation.
- Cost and risk high. But hope of interest for further research.
- Demonstrates the power and flexibility of deep learning approaches…
- Broadly we have trained faster (concurrent & differentiable) proxy models. If you like this work then there are other applications:
  - Activity schedule synthesis/generation
  - Influence on behavioural theory
  - Latent representations can be used to measure distances

# Further Work

- Activity schedule synthesis:
  - Synthetic datasets
  - Upsampling for simulations
  - Modelling as part of an activity-based model
- Increase model capabilities (trips, modes, multi-days).
- Identify and disseminate best performing encodings and architectures (CAVEAT).
- Apply to a multi-agent simulation optimisation problem - EV charging?

**Fred Shone** - frederick.shone.17@ucl.ac.uk

Dr Hillel - UCL BIG - tim.hillel@ucl.ac.uk

Dr Perez-Ortiz - UCL AI Centre

https://fredshone.github.io/

frederick.shone.17@ucl.ac.uk

github.com/fredshone

Thank you Arup City Modelling Group!

DCM extension - github.com/matsim-org/matsim-libs/tree/master/contribs/discrete_mode_choice

CAVEAT - github/fredshone/caveat

PAM - github/arup-group/pam

big
behaviour & infrastructure group

Engineering and Physical Sciences Research Council