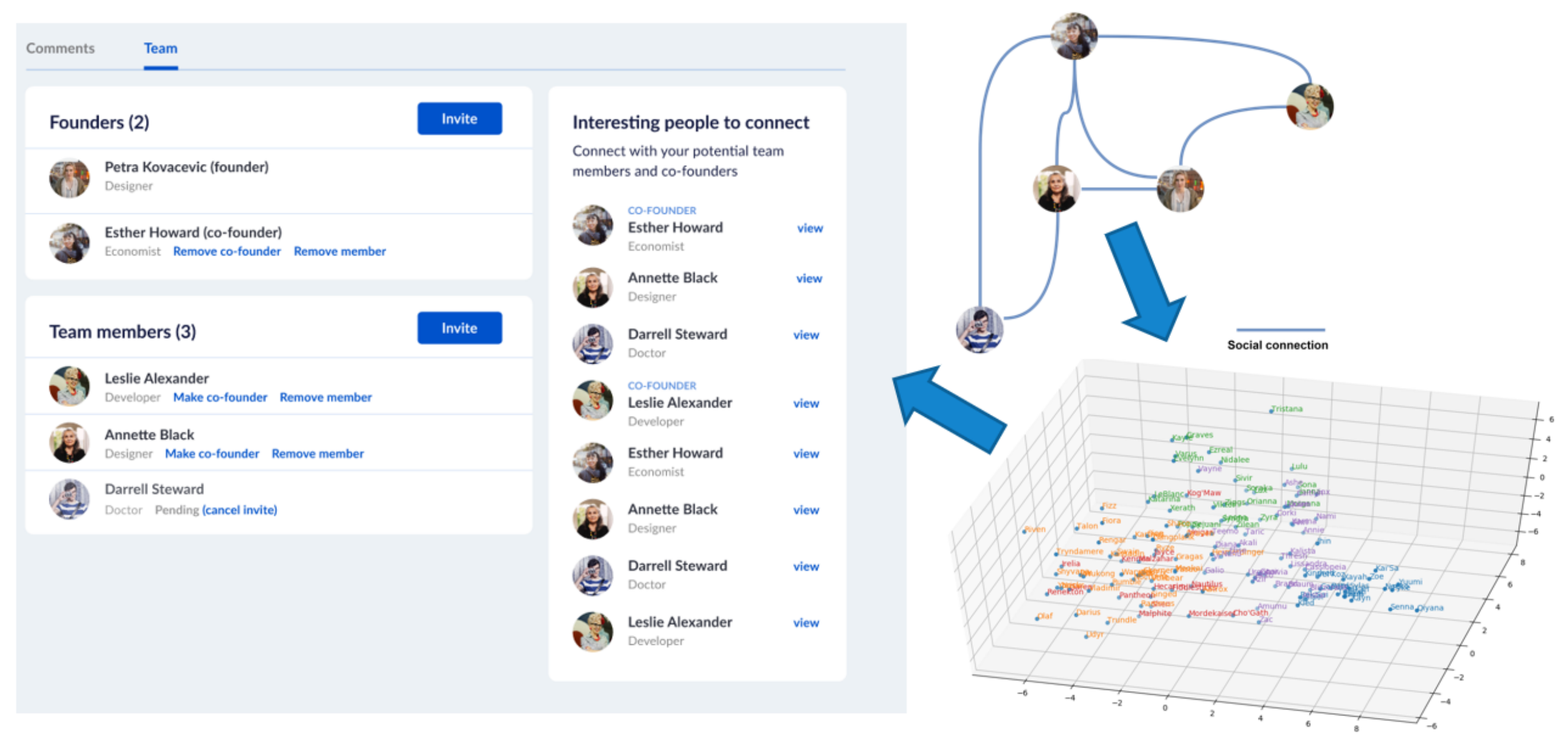
Recommendations in a Multi-Domain Setting: Adapting for Customization, Scalability and Real-Time Performance

Emanuel Lacic
elacic@know-center.at

Dominik Kowald
dkowald@know-center.at

Multi-Domain Support

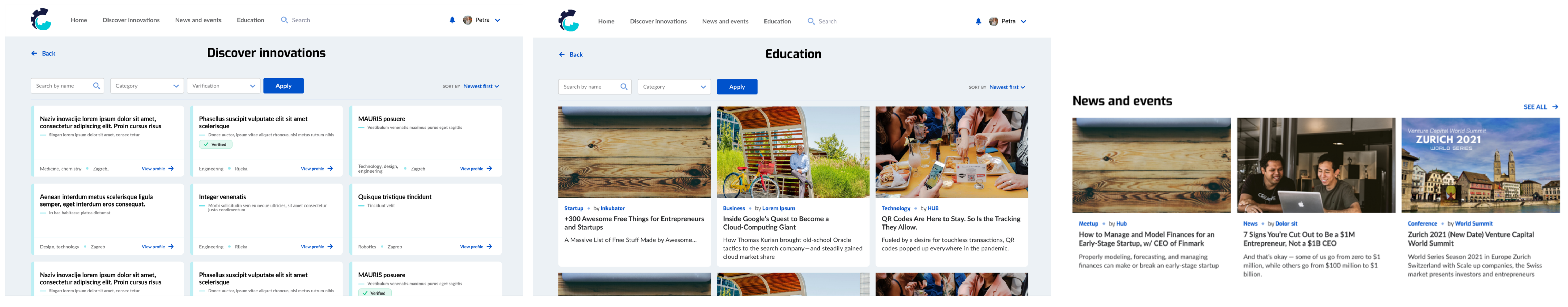
Example: Entrepreneurial start-up founding. Such a setting entails diverse personalization scenarios. A recommender system needs to suggest (i) relevant experts that can provide feedback to an innovation idea, (ii) support potential co-founder and team member matching, (iii) allow accelerators, incubators, and innovation hubs to discover these innovations as well as (iv) continuously provide relevant education materials until the innovation idea has become mature enough in order to form a start-up.



Objective: The utilized recommender algorithms need to be customized with respect to the underlying data structures.

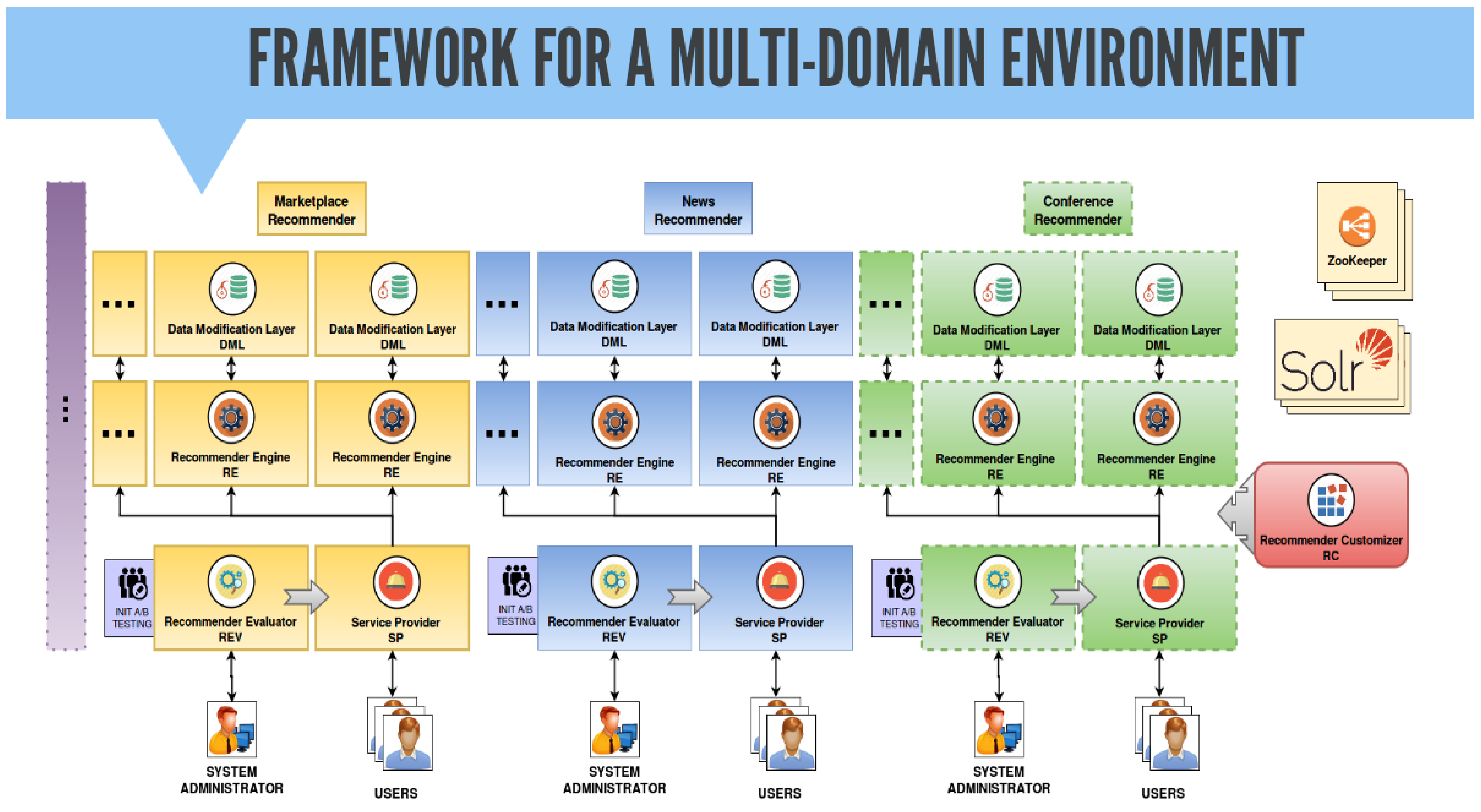
Co-Founder Matching:

- Increase skill diversity
- Maintain personal compatibility



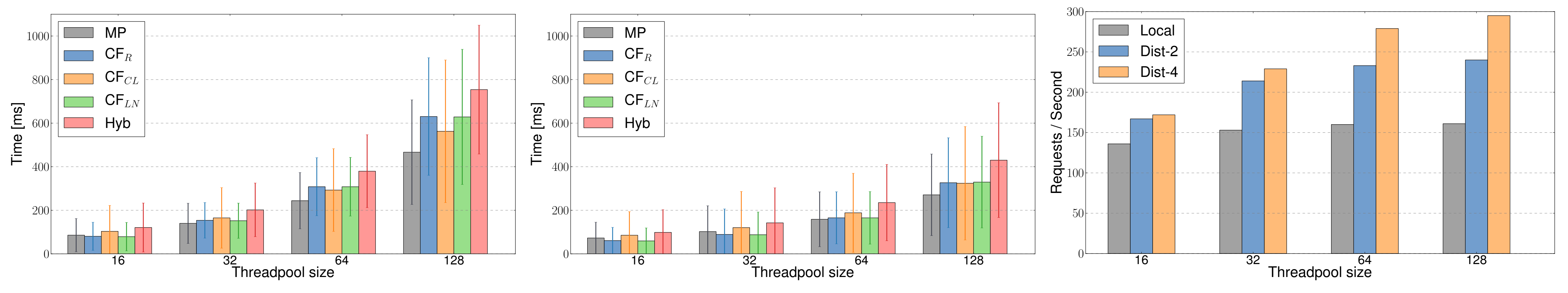
Production Setup. Different challenges need to be addressed in order to continuously maintain the stability and health of a recommender system. A distributed architecture should be guided by design principles that:

- Provide service isolation
- Support data heterogeneity
- Allow for algorithmic customization
- Ensure fault tolerance



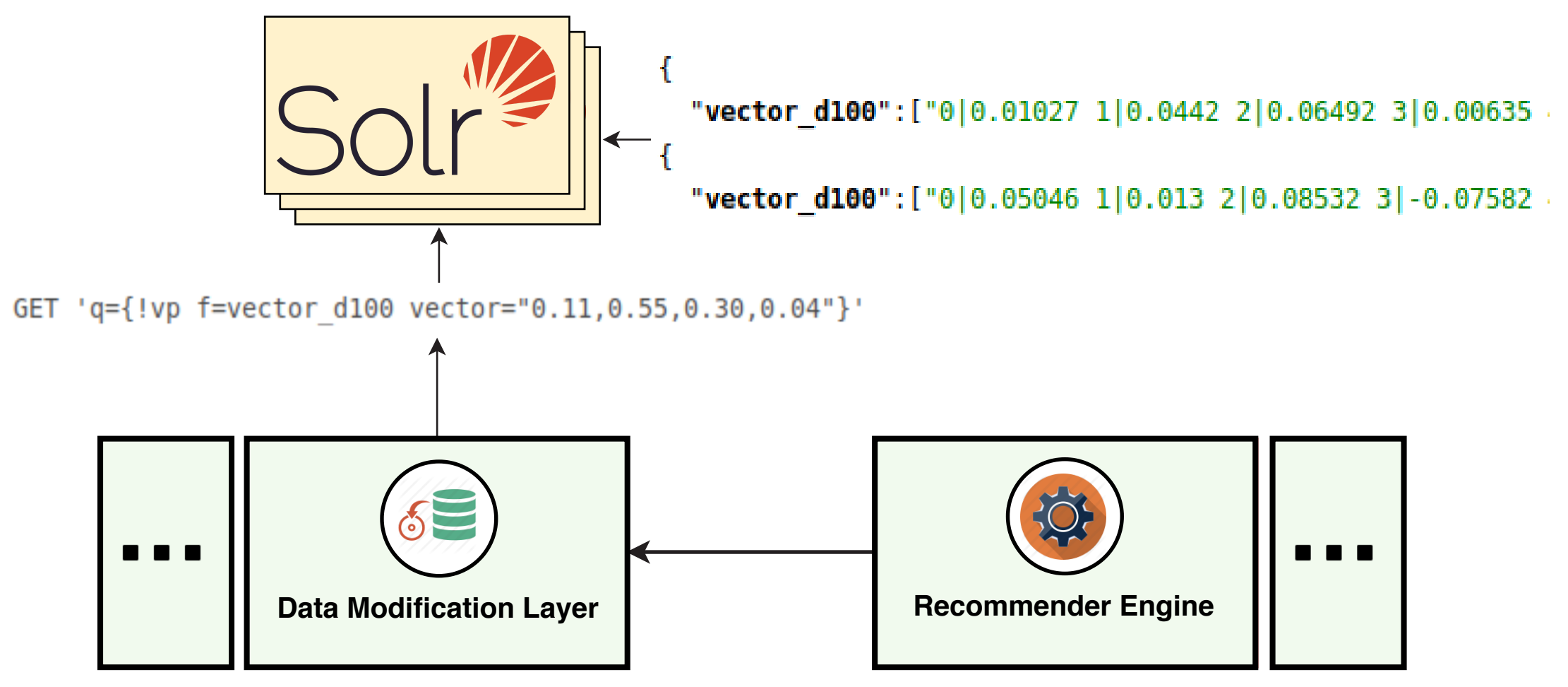
Scalability

Load Test. A threadpool simultaneously requests recommendations. If we scale-up the **number of processing nodes** (e.g., from 1 to 4), we can guarantee a much higher number of **recommendation requests per second** while maintaining the **desired runtime** performance.



Real-Time Performance

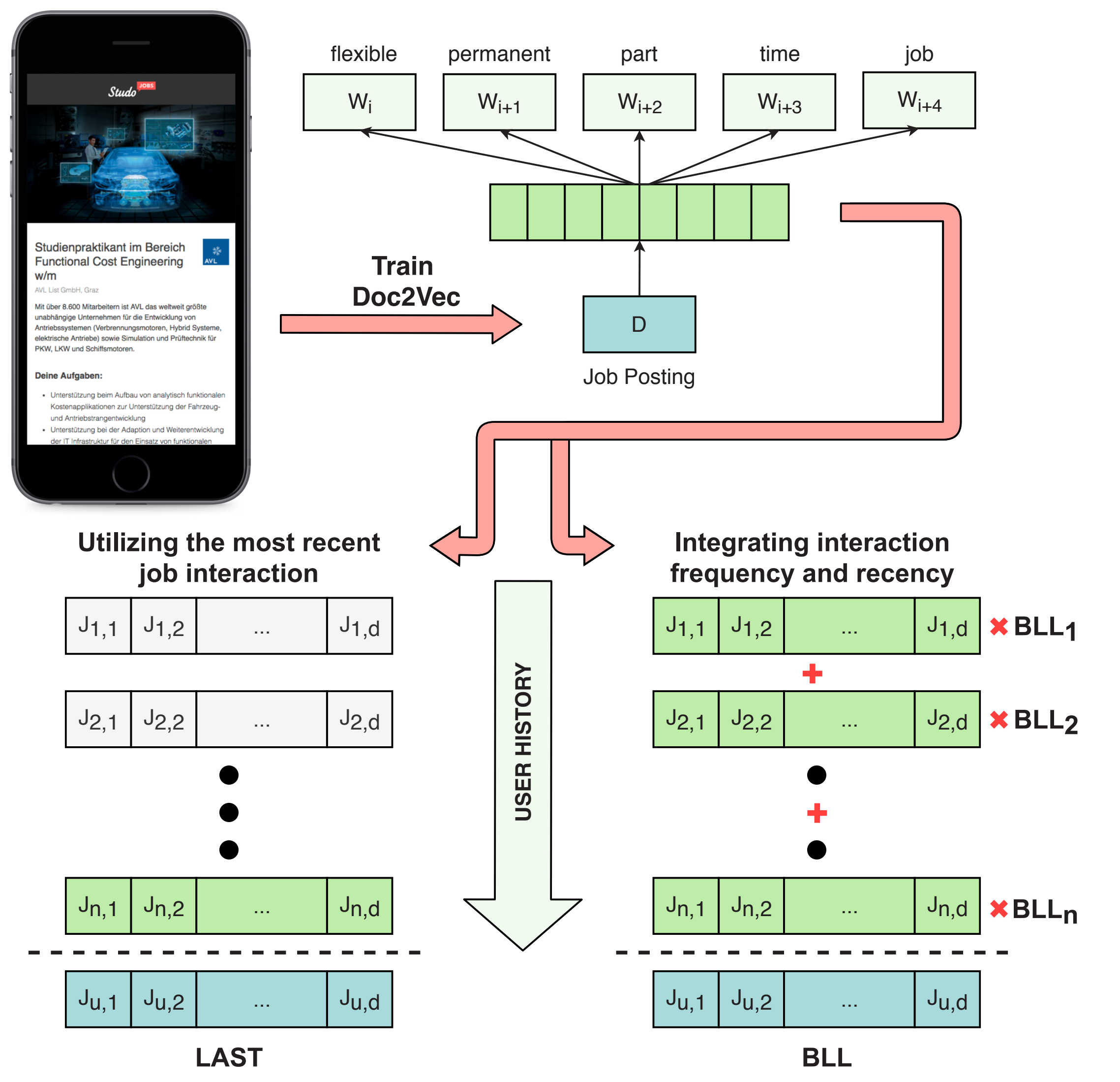
Adapting for real-time recommendations. Response times in most cases need to be below 100-200 milliseconds. Embeddings can be stored as payloads in Lucene and Cosine similarity can be used for retrieval.



```
{
  "vector_d100":["0|0.01027 1|0.0442 2|0.06492 3|0.00635
{
  "vector_d100":["0|0.05046 1|0.013 2|0.08532 3|-0.07582
```

```
GET 'q={!vp f=vector_d100 vector="0.11,0.55,0.30,0.04"}'
```

Example: Job Marketplace. Stored embeddings can be combined in real-time in order to optimize between favoring frequency or recency of consumed content.



flexible
permanent
part
time
job
W_i
W_{i+1}
W_{i+2}
W_{i+3}
W_{i+4}
Train Doc2Vec
D
Job Posting
Utilizing the most recent job interaction
Integrating interaction frequency and recency
USER HISTORY
$J_{1,1}$ $J_{1,2}$ ... $J_{1,d}$
$J_{2,1}$ $J_{2,2}$ ... $J_{2,d}$
$J_{n,1}$ $J_{n,2}$ ... $J_{n,d}$
$J_{u,1}$ $J_{u,2}$ ... $J_{u,d}$
$\times BLL_1$
$\times BLL_2$
$\times BLL_n$
LAST
BLL

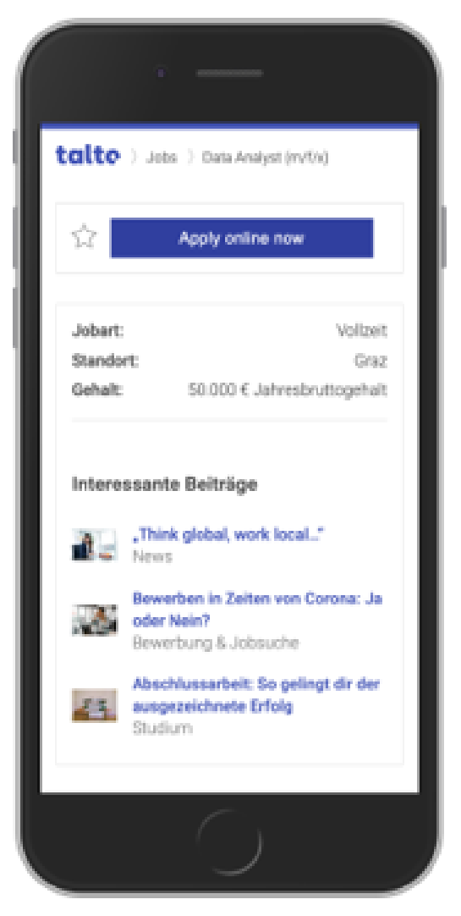
BLL. Accounts simultaneously for both **frequency** and **recency**. Browsing behaviour for a reference vector is modelled by:

$$BLL_{u,j} = \ln\left(\sum_{i=1}^{n} (TS_{ref} - TS_{j,i})^{-d}\right)$$

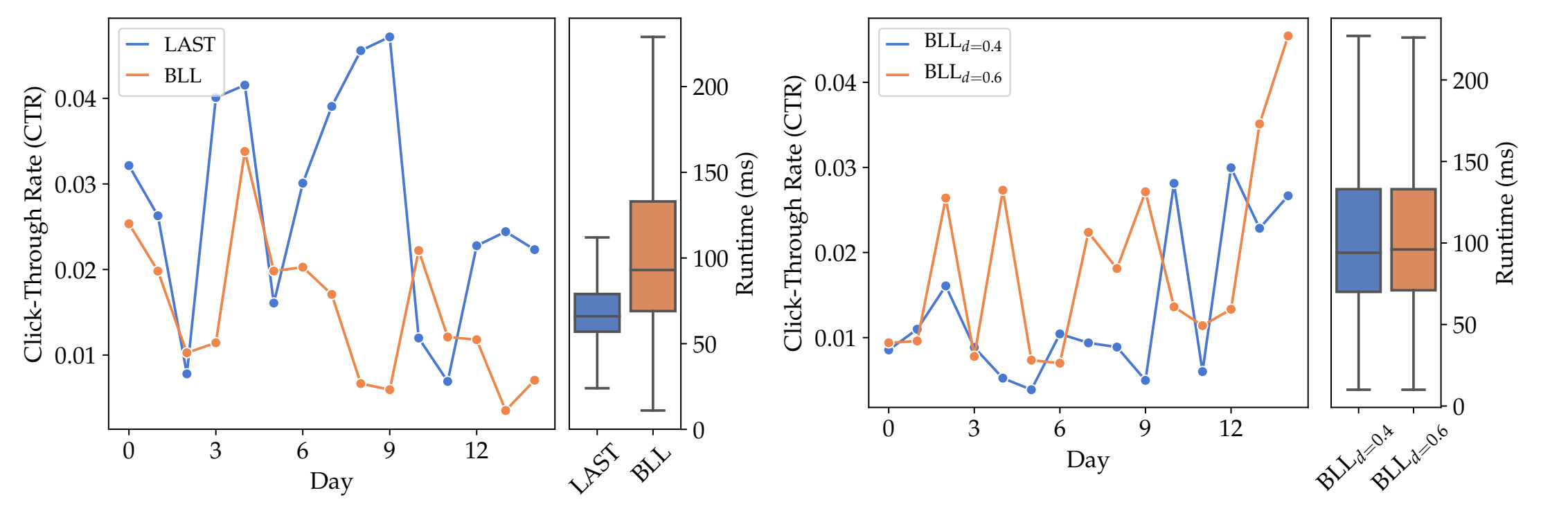
where d represents the time-dependent decay parameter.

Customization

There is no one-size-fits-all solution. Domain-specific algorithm configurations need to be set up and evaluated accordingly.



```
!!at.knowcenter.sc.recomm.common.profiles.VectorCbProfile
recommType: items
fetchUserHistoryProperties:
  itemCoreProperties:
    interactionFields:
      fields:
      - name: users_viewed
      - name: users_applied
      - name: users_showed_contact
      - name: users_viewed_company
      - name: users_shared
vectorProperties:
  similarityType: cosine #euclid, dotproduct
  mergeStrategy: bll
  timeExponentD: -0.5 # d exponent value in the BLL equation
  vectorField: vector_d100
similarItemsProperties:
  filterClauses:
    clauses:
    - connector: AND
      filters:
      - field: status
        values:
        - PUBLIC
    connector: AND
    minus: false
```


LAST
BLL
Click-Through Rate (CTR)
Day
Runtime (ms)
$BLL_{d=0.4}$
$BLL_{d=0.6}$