THE APPLICATION OF REINFORCED MACHINE LEARNING FOR CACHE MANAGEMENT IN HIGH-LOAD INFORMATION SYSTEMS

Authors:
Bohdan Volokh (Speaker)
Kyiv National University of Construction and Architecture, Ukraine

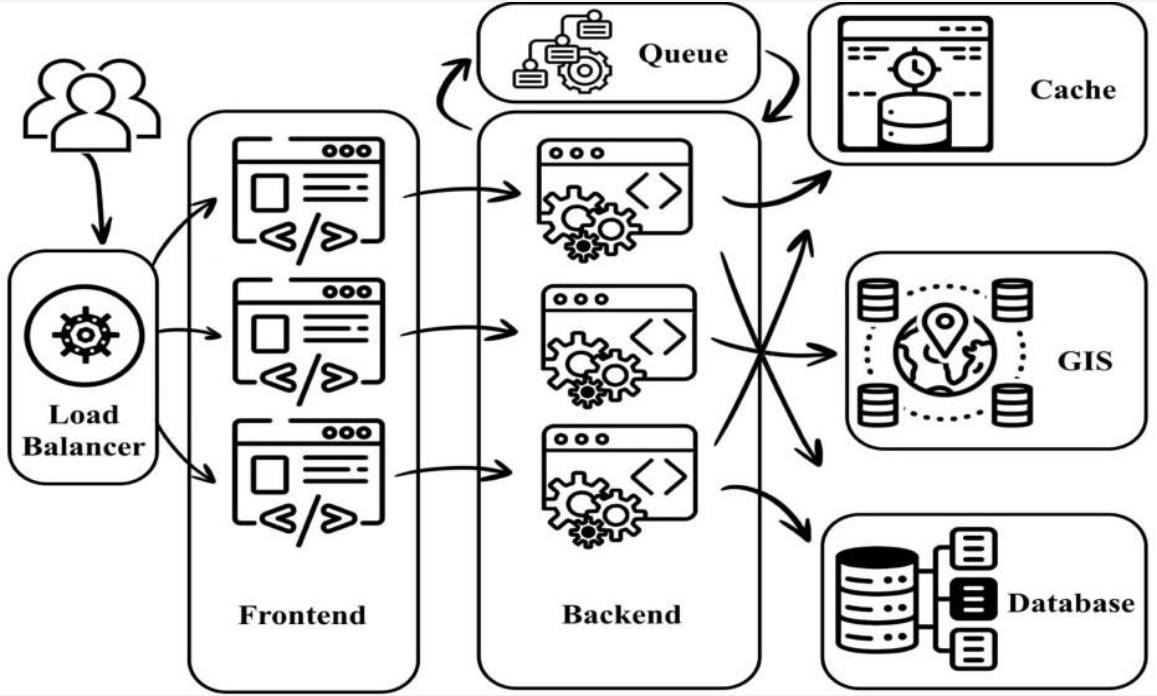
Svitlana Terenchuk
Professor
Kyiv National University of Construction and Architecture, Ukraine

Bohdan Yeremenko
Associate Professor
Taras Shevchenko National University of Kyiv, Ukraine

Background: Consequences of Recent Missile Attack on Kyiv

Initially, the developed approach was intended for a high-load informational **System supporting the restoration of damaged real estate objects (System)**. However, today there is an urgent need for efficient high-load information systems in healthcare, construction, and logistics.

Initial architecture of high-load informational System [1]



S. Terenchuk, R. Pasko, A. Buhrov, V. Ploskyi, O. Panko and V. Zapryvoda, "Computerization of the process of reconstruction of damaged or destroyed real estate," 2022 IEEE 3rd KhPI Week on Advanced Technology (KhPIWeek), Kharkiv, Ukraine, 2022, pp. 1-6, https://doi.org/10.1109/KhPIWeek57572.2022.9916470

From Domain-Specific to Domain-Independent Caching

The proposed approach was initially developed for a system supporting the restoration of damaged real estate objects, but it is domain-independent and can be applied to other high-load systems.

Original Domain

Damaged Real Estate Restoration System

- Property objects (buildings, apartments)
- Damage characteristics
- Restoration status
- User roles (citizens, contractors, government)
- Request patterns and priorities

Domain-specific state features

Reinforcement Learning Cache Management

(Q-Learning)

State (object & user features) → **Action** (admit / reject) → **Deferred Reward** (future cache hits)

Q-Update

The decision-making mechanism remains unchanged.

Only the state features are adapted to the data and usage context of a particular domain.

Other Application Domains

Healthcare (e.g., CDSS)

- Patient data
- Medical records and recommendations
- High-load clinical queries

Construction

- Project data
- Resource allocation
- Document management

Logistics

- Shipments and routes
- Real-time tracking data
- High-frequency location queries

Adapted domain-specific state features

Same decision-making mechanism. Different domain-specific state features.

Relevance and Research Aim

The relevance of the research is determined by the need to improve data access efficiency in modern high-load information systems.

The aim of the research is to develop an adaptive approach for making efficient decisions on adding new objects to the cache based on reinforcement learning.

Object and Subject of the Research

The relevance of the research is determined by the need to improve data access efficiency in modern high-load information systems.

The aim of the research is to develop an adaptive approach for making efficient decisions on adding new objects to the cache based on reinforcement learning.

Data Processing Problems Addressed by Caching and NoSQL Databases

Problem	Possible Solution
Data are stored in different formats	Flexible data schema and JSON support (NoSQL database)
No single data source	Centralized storage with horizontal scaling (NoSQL database)
Poor performance under a large number of requests	Horizontal database scaling + caching of frequently requested data
Object popularity and relevance are not considered	Intelligent caching based on request dynamics and history
Repeated processing of the same requests	Caching query results and reducing database load

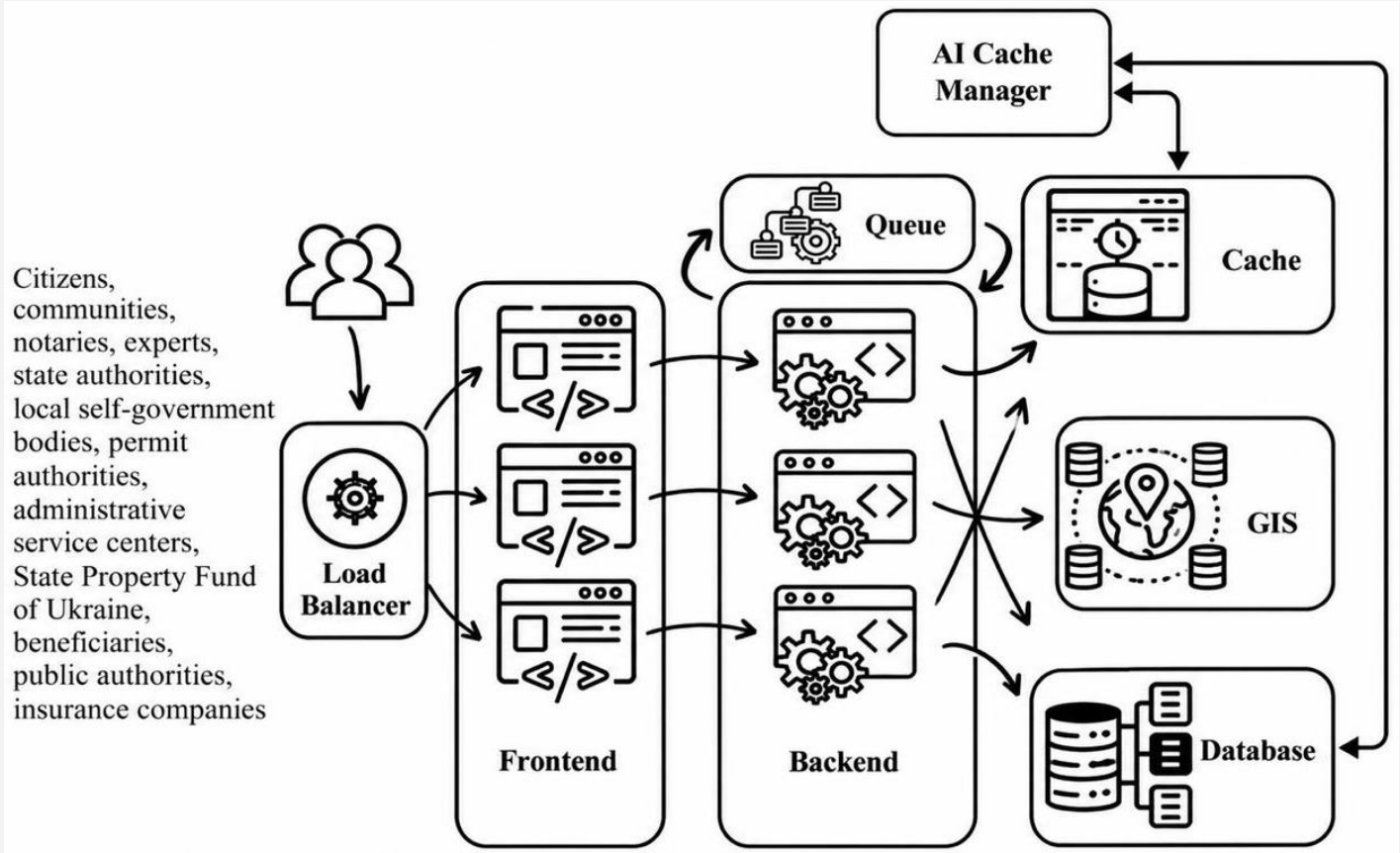
Why is intelligent caching needed?

Caching is a technology for accelerating data access by temporarily storing frequently used information in a high-speed intermediate storage – the cache.

This reduces the load on the primary database and accelerates the processing of repeated requests.

A cache management method is a strategy that determines:
– which data should be added to the cache;
– which data should be removed when the cache is full;
– when and how the cache contents should be updated.

Proposed Architecture of the System with Intelligent Caching



Intelligent Caching Problem Statement

The goal of intelligent caching is to make effective decisions about storing data in the cache based on request history and key object features.

Therefore, Intelligent Caching Approach requires solving an important problem:
Should a new object be added to the cache?

Problem Formalization

The cache admission decision is formalized as a Markov Decision Process (MDP):

$$M = \langle S, A, P, R, \gamma \rangle$$

where:

- S – the set of states, represented by the object feature vector $x = [x_1, x_2, \ldots, x_n]$;
- A = {0,1} – the set of actions: 0 – do not cache, 1 – cache;
- P(s′|s,a) – probability of transition to a new state s′ after action a in state s;
- R(s,a) – reward function for performing action a in state s;
- $\gamma \in [0,1]$ – discount factor representing the importance of future decisions.

Goal – find an optimal policy $\pi^*(s)$ that makes cache-admission decisions with maximum long-term utility for the system.

Comparison of Caching Methods

Efficiency and Implementation Complexity

Caching Method	Decision Time	Training or Update Time	Implementation Complexity	Adaptability to Changes	Decision Accuracy
LRU	**Very fast** (O(1))	Not required	Low	✗ No	Low
LFU	**Fast** (O(log(n)), n – number of objects in cache)	Not required	Medium	✗ No	Moderate
Tabular Q-Learning	**Fast** (O(1))	Moderate	Medium	✓ Yes	High

Machine learning-based approaches, such as tabular Q-Learning, provide higher adaptability and decision accuracy compared to traditional heuristic methods, at the cost of increased complexity.

Q-Function

The Q-function value is updated using the Bellman equation:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r_t + \gamma \cdot \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t) \right]$$

where

- $Q(s,a)$ – utility function;
- s_t – current state (object feature vector);
- a_t – selected action (0 or 1);
- r_t – reward obtained after performing the action;
- s_{t+1} – next state after the action;
- $\alpha \in (0,1]$ – learning rate;
- $\gamma \in [0,1]$ – discount factor;
- $\max_{a'} Q(s_{t+1},a')$ – estimate of the best future action.

Q-Table Collection

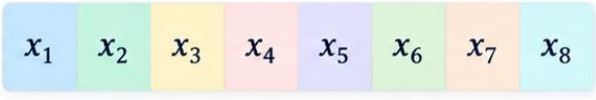
Field	Data Type	Description
state	Object or String (JSON)	Composite state representation – feature vector
action	Integer (Int)	Action: 0 – do not cache; 1 – cache
q-value	Float	Q(s,a): expected reward for performing the action

It is stored separately in MongoDB NoSQL database and is used by the tabular Q-Learning method.

Feature Vector Used to Represent the State in the Model

Notation	Feature Name	Description
x_1	**Number of unique users**	The number of different users who accessed the object within the last 24 hours.
x_2	**Total number of requests**	The total number of requests to the object during the last 24 hours.
x_3	**Object type**	Type of the object (e.g., 0 – residential, 1 – school, 2 – hospital, etc.).
x_4	**Hours since last request**	The number of hours since the last request, reflecting the recency of interaction.
x_5	**Object or area priority**	Priority level of the object or area (e.g., 0 – low, 1 – medium, 2 – high).
x_6	**Object location**	Categorical feature of the object's district/region or its encoded value.
x_7	**User role**	Categorical feature representing the type of user who made the request (e.g., citizen, expert, authority).
x_8	**Time of day**	Indicates the time period of the last request (e.g., morning, daytime, evening, night).

State Representation (Feature Vector)



$$s = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$$

The feature vector combines usage statistics, object characteristics and contextual information about users and time to represent the state of an object in the caching model.

Reward Definition

No.	Situation	Reward $r^{(t)}$	Explanation
1	Object was cached and requested again	+1	Caching was justified: the object was reused.
2	Object was cached but was not requested again	−1	Caching was unnecessary: the object occupied cache space but was not reused.
3	Object was not cached and no request followed	0	Neutral case: nothing was lost and nothing was gained.
4	Object was not cached but was requested later	−1	A potential cache hit was lost: the object could have been useful.

Deferred Asynchronous Q-Value Update Mechanism

After each cache-admission decision, information about the action is stored in a special list.
Each its record contains:

- s – object state represented by the feature vector;
- a – selected action (0 or 1);
- object_id – object identifier used for subsequent analysis;
- t – action timestamp.

After a fixed interval T, the method checks whether the object was requested again and determines the reward. The corresponding Q-value is then updated asynchronously, without blocking request processing.

Action Selection – ε-Greedy Strategy

◆ To prevent learning from getting stuck on a single decision, an ε-greedy strategy is used:
- with probability ε: a random action is selected (exploration);
- with probability 1−ε: the optimal action is selected (maximum Q(s,a)).

◆ At the beginning of training, ε_t is set high (for example, 0.9) to explore possible states. It then decreases exponentially:

$$\varepsilon_t = \max(\varepsilon_{min}, \varepsilon_0 \cdot e^{-kt})$$

where:
- ε_0 – initial value;
- ε_{min} – minimum allowed value;
- k – decay-rate coefficient;
- t – training step or episode number.

◆ **This strategy balances exploration of new actions with exploitation of already learned optimal decisions.**

Algorithm

1. Initialization:

   Create an empty Q-table.

   Set the parameters:

   α – learning rate;

   γ – discount factor;

   ε_0, ε_{min}, k – ε-greedy strategy parameters;

   T – waiting time for the action outcome.

2. Request Processing (time t):

   Form the object state $s^{(t)}$.

   Select action $a^{(t)}$ using the ε-greedy strategy.

   Perform the action (e.g., cache the object).

   Save a record to the list: ($s^{(t)}$, $a^{(t)}$, object_id, t).

Algorithm

3. After time T:
   Retrieve from the list the record whose timestamp corresponds to the required update time;
   Using object_id, form the new state $s^{(t+T)}$ from the current object features;
   Determine reward r based on the actual subsequent use of the object;
   Update the corresponding record in the Q-table.

4. Update ε:
   with probability ε: select random action;
   with probability 1−ε: select the action with the maximum Q(s,a).

Method Operation Summary

• The tabular Q-Learning method learns from experience by accumulating information about actions and their consequences.

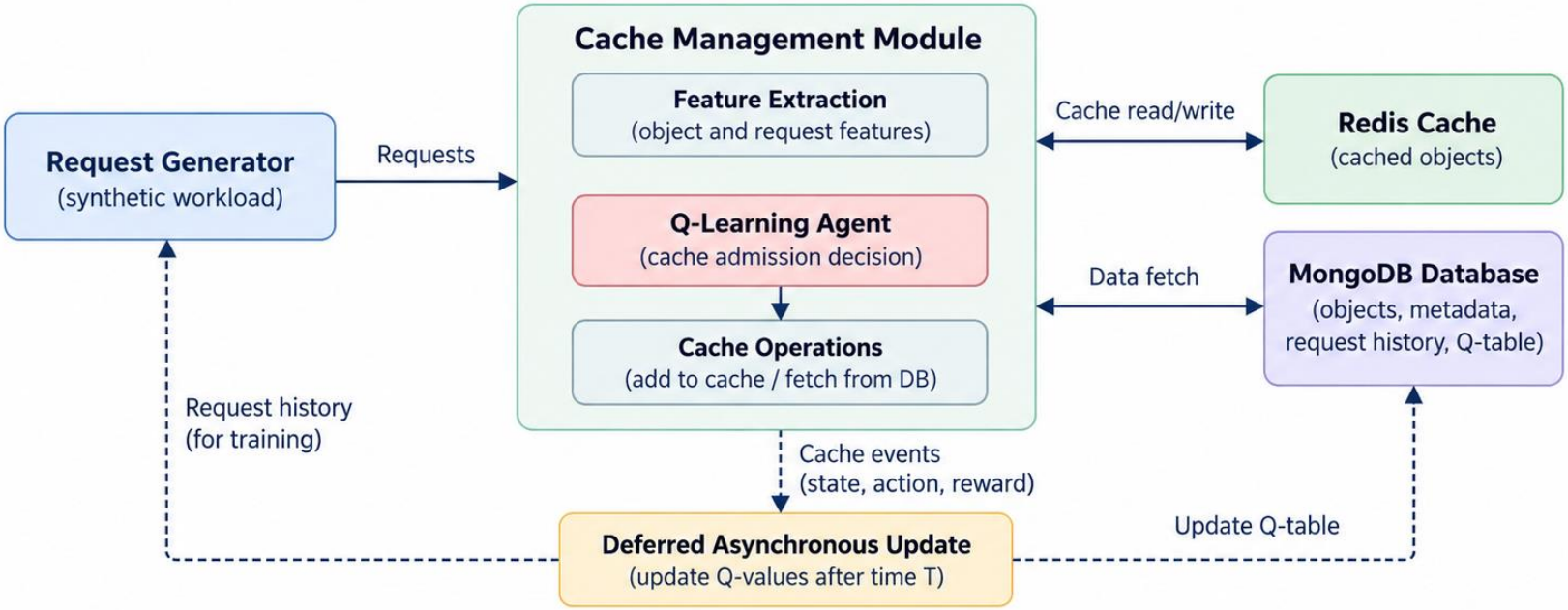
• With deferred updates, the outcomes of cache-admission decisions are evaluated after time T.

• After training, the Q-table is used as the decision-making policy for cache admission.

• If user behavior or request patterns change, the method can be retrained to adapt the caching policy.

Implementation

Experimental system for testing the proposed approach



- **Python** – implementation of the experimental system
- **MongoDB** – storage of object data, request history and Q-table
- **Redis** – caching technology
- **Q-Learning Agent** – decision-making on whether a new object should be added to the cache
- **Deferred Asynchronous Update** – updating Q-values based on the observed result after time T

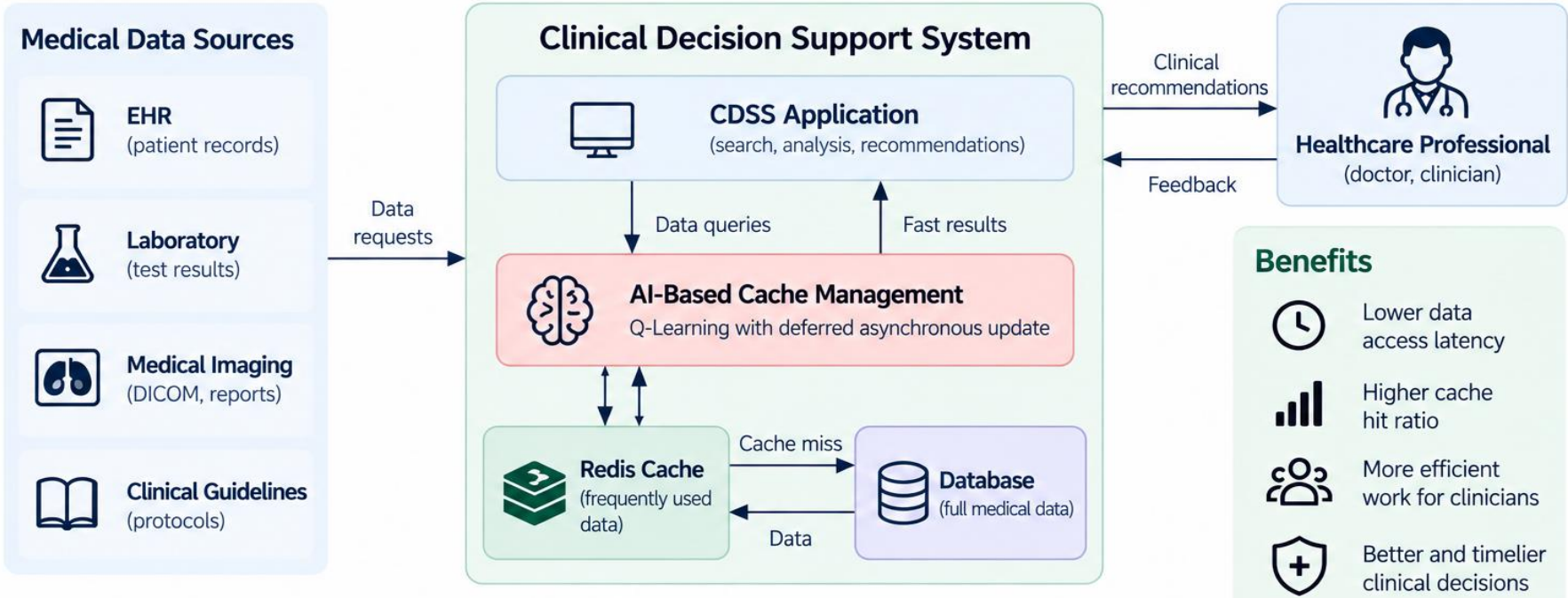
Accumulated Learning Across Multiple Runs

The Q-table is preserved between runs, allowing the method to accumulate experience and progressively improve its cache-admission policy.

The Q-table is preserved between consecutive runs, allowing the method to accumulate experience and progressively adapt its cache-admission policy.

Potential Application in a Clinical Decision Support System

Using intelligent caching to provide faster access to up-to-date medical data



The proposed caching approach can improve the efficiency of CDSS users by providing faster access to current medical data, leading to better-informed clinical decisions.

Conclusions

- The cache admission problem was formulated as a sequential decision-making problem and solved using the tabular Q-Learning method.
- The proposed approach takes into account object characteristics and request context when deciding whether a new object should be added to the cache.
- A deferred asynchronous update mechanism was developed to evaluate decisions based on their long-term consequences and update Q-values without blocking request processing.
- The proposed approach provides adaptive cache-admission decisions and can be applied to other high-load information systems, including Clinical Decision Support Systems, by adapting the state features to the characteristics of the data and their usage context.

Thank you for your attention!

Dr. Bohdan Volokh, PhD
Senior Instructor
Kyiv National University of Construction and Architecture
- linkedin.com/in/bohdan-volokh