HyperSeq

A Hyper-Adaptive Representation for Predictive Sequencing of States

Koohestani, Roham; Izadi, Maliheh

DOI
10.1145/3696630.3728526

Publication date
2025

Document Version
Final published version

Published in
FSE Companion 2025 - Companion Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering

Citation (APA)
Koohestani, R., & Izadi, M. (2025). HyperSeq: A Hyper-Adaptive Representation for Predictive Sequencing of States. In J. Li (Ed.), *FSE Companion 2025 - Companion Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering* (pp. 696-700). (Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering). ACM. https://doi.org/10.1145/3696630.3728526

HyperSeq: A Hyper-Adaptive Representation for Predictive Sequencing of States

Roham Koohestani
Delft University of Technology
Delft, The Netherlands
rkoohestani@tudelft.nl

Maliheh Izadi
Delft University of Technology
Delft, The Netherlands
m.izadi@tudelft.nl

Abstract

In the rapidly evolving world of software development, the surge in developers' reliance on AI-driven tools has transformed Integrated Development Environments into powerhouses of advanced features. This transformation, while boosting developers' productivity to unprecedented levels, comes with a catch: increased hardware demands for software development. Moreover, the significant economic and environmental toll of using these sophisticated models necessitates mechanisms that reduce unnecessary computational burdens. We propose HyperSeq — Hyper-Adaptive Representation for Predictive Sequencing of States — a novel, resource-efficient approach designed to model developers' cognitive states. HyperSeq facilitates precise action sequencing and enables real-time learning of user behavior. Our preliminary results show how HyperSeq excels in forecasting action sequences and achieves remarkable prediction accuracies that go beyond 70%. Notably, the model's online-learning capability allows it to substantially enhance its predictive accuracy in a majority of cases and increases its capability in forecasting next user actions with sufficient iterations for adaptation. Ultimately, our objective is to harness these predictions to refine and elevate the user experience dynamically within the IDE.

CCS Concepts

• **Human-centered computing**; • **Software and its engineering**;

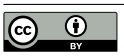
Keywords

User Behavior Modeling, IDE Design, Formal Methods

ACM Reference Format:
Roham Koohestani and Maliheh Izadi. 2025. HyperSeq: A Hyper-Adaptive Representation for Predictive Sequencing of States. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23–28, 2025, Trondheim, Norway*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3696630.3728526

1 Introduction

Since the advent of artificial intelligence, the software development life-cycle has become increasingly entangled with AI tooling. The introduction of tools like Copilot [4] and JetBrains AI [5] is enhancing user productivity, while editors like Cursor [3] simplify programming, thereby transforming the way developers engage with software repositories.



FSE Companion '25, June 23–28, 2025, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/2025/06
https://doi.org/10.1145/3696630.3728526

Together with these benefits, it should be noted that the underlying models have added complexity to the systems we program, consequently raising the computational demands of programming. This increased computation demand has led to greater environmental repercussions; a recent technical blog by the team responsible for the Llama 3.1 model series reveals that training the leading 405 billion parameter model consumed 30.84 million GPU hours [1]. This translates into 7.92 kilotons of CO2 emissions, based on the setup detailed in the article [11] and using the United States emission factor of 367 grams of CO_2 per kWh [2]. Furthermore, executing inferences with this model demands a multi-node arrangement, with each node equipped with 8 H100 GPUs operating at a 700W TDP, highlighting that a substantial impact of these models occurs during the post-deployment phase.

Previous studies have attempted to leverage user telemetry data to model user behavior [15] and to filter out completion requests in the IDE by taking into account the user's present state [8, 16]. Furthermore, earlier research has differentiated between two categories of user states: exploration, where a user seeks suggestions, and acceleration, where the user aims to continue programming without interruption. However, these existing methods are still rather costly to train and require a large amount of training data.

Moreover, customizing models for individual users is challenging because a single model cannot adequately capture all users' behaviors. Adapting a model to a user's behavior requires ongoing online learning and efficient implementation.

Hyperdimensional computing (HDC)[13] provides a framework for holistic representation, manipulation, and querying of abstract concepts. This approach encodes ideas within high-dimensional vector spaces, taking advantage of the quasi-orthogonality inherent in these vectors.

This paper presents HyperSeq, a Hyper-Adaptive Representation for Predictive Sequencing of States. We evaluate its effectiveness in a set of adaptive and non-adaptive scenarios using real-world user interaction data. The model predicts user states with over 70% accuracy, and adaptation mode significantly boosts accuracy through online learning.

In this paper, we (1) establish the theoretical basis for a hyperdimensional sequence model, (2) formally define predictive sequencing and demonstrate that the model allows for training in $O(D \cdot |Train|)$, and online adaptation in $O(D)$ time, (3) conduct an initial assessment of the model's performance, and (4) outline potential future applications for HyperSeq.

2 Motivation

There is a tremendous amount of cost associated with running these large models, especially when dealing with large-scale inference. An

approach to addressing this issue involves minimizing the number of incorrect invocations, meaning decreasing the volume of calls that are likely to result in the rejection of generated responses. By analyzing user states, we can determine the likelihood of a user accepting or rejecting completions, thus alleviating server strain.

Moreover, recent research indicates [19] that developers prefer to incorporate proactivity within their systems. By forecasting upcoming states and user actions, we can strive to embed state-aware mechanisms into IDEs.

For example, detecting when a user is about to enter refactoring mode could allow the IDE to proactively initiate a search for potential refactorings and suggest these options as soon as the mode change is detected.

3 Background Literature

3.1 Hyper-dimensional Computing

Hyper-Dimensional Computing (HDC) is a computational paradigm in which high-dimensional vectors, typically ranging from 10^4 to 10^5, are utilized to represent, process and query abstract concepts. Although HDC was coined by Pentti Kanerva [13], the concept dates to the 1990s, with holographic reduced representation (HRR) [18] and Vector-Symbolic Architectures [10].

In recent years, there has been growing interest in employing HDC for intelligent tasks because of the efficiency of models based on HDC. Such HDC-based methods have been applied to challenging problems like Raven's Progressive Matrices [12], as well as in encoding visual representations efficiently and retrieving this data using what is referred to as resonator networks [14]. Recent studies have explored the integration of HDC-inspired modifications into current architectures. InfiniAttention [17] has demonstrated significant advancements in processing exceptionally lengthy contexts by encoding them into a sparse hyperdimensional memory.

3.2 User-IDE interaction

Prior research has developed taxonomies of user interactions in IDEs, focusing on AI features. One study [7] identified two modes: exploration, where developers seek recommendations and solutions, and acceleration, where they work confidently with minimal distractions.

Various investigations have employed telemetry data from within the IDE to determine programmers' conditions. By analyzing features like caret movements and typing speed, these studies inferred the action states and developed a classification of 12 user states, along with their transitional characteristics [15]. The concept of leveraging telemetry data to enhance Human-AI interaction remains prevalent, with numerous studies applying it to effectively filter out requests that are highly likely to be declined [8, 16].

Researchers have examined how to design Human-AI eXperiences (HAX) within the IDE by establishing a classification of methods through which developers seek assistance and identifying various domains where distinct models can provide support, including testing, maintenance, and optimization.

3.3 Next-action prediction

So far, limited research has focused on predicting users' next actions within IDEs. Cursor, a newly developed AI-integrated IDE, frequently references next-action prediction on its website, suggesting significant investment in this area [3]. Moreover, existing studies have explored employing gaze-tracking to dynamically adjust the IDE according to the user's forthcoming actions [20].

4 Problem Definition

In this section, we formally define the problem statement to improve clarity for subsequent sections. As noted in section 8, our present work concentrates on predicting future user states given the limited availability of data. Nevertheless, we plan to broaden the investigation in future research to deduce subsequent user states directly from user telemetry.

4.1 Predictive Sequencing

Let S be the set containing all potential states, and let s indicate an individual state within S. Similarly, let U signify the set of all users, with u representing a single user in U. A session state sequence, denoted $Sess$, is defined as an m-tuple of states $Sess = (s_0, s_1, \ldots, s_m)$, where each $s_i \in S$. Furthermore, we denote the i-th session of user u as $Sess_{u,i}$. The objective of next-state prediction is to develop a model M capable of forecasting the following state for a user u based on an n-tuple of states. Formally, the model is articulated as: $M : S^n \mapsto S$. Additionally, we define a user-specific model as M_u, tailored for the individual user u.

4.2 Hyper-Dimensional Computing (HDC)

In this section, we expand upon the formal structure introduced earlier to lay the theoretical groundwork for our approach. We adhere to the following notational conventions:

- The bundling operator is represented by $\oplus$,
- The binding operator is signified by $\otimes$, and
- The permutation operator is symbolized by P.

Further, we utilize the Bipolar Map framework, wherein hypervectors are drawn from the domain $DOM = \{-1, 1\}^D$, with D denoting the dimensionality of the hyperspace. The elements of the chosen hypervectors are sampled from the probability distribution $P = 2 \cdot (\text{Ber}(0.5) - 0.5)$ [9].

A hypervector v is characterized as $v \in DOM$. A codebook C links the state space S to the hyperspace DOM, i.e., $C : S \mapsto DOM$. Likewise, C' provides the reverse mapping, defined as $C' : DOM \mapsto S$. The function $Encode$ is designed to process an n-tuple of hypervectors v_i while maintaining their sequential characteristics. Formally, $Encode : DOM^m \mapsto DOM$, and it is implemented as following where $V = (v_0, v_1, \ldots, v_{n-1})$,

$$Encode(V) = \bigotimes_{i=0}^{n-1} (P^{n-i-1}(v_i)).$$

Additionally, we define a function Sim that takes two inputs from the defined hyperspace and yields a real-number indicating the similarity between the two vectors. More formally, $Sim : (DOM, DOM) \mapsto \mathbb{R}$. In our context, we measure similarity using cosine similarity, with values ranging from -1 to 1, where -1 represents complete dissimilarity and 1 represents complete similarity.

5 HyperSeq

5.1 Representation

As stated earlier, the challenge of depicting action states involves encoding every action subsequence of length n and then using the model to predict the most probable action from a subsequence of $n-1$ actions. Hence, a foundational model M_{Base} can be developed utilizing the training dataset $Train$

$$\bigoplus_{i=0}^{|Train|-1} (Encode(Seq_i))$$

where

$$Train = \{Seq_0, Seq_1, \cdots, Seq_{|Train|-1}\}$$
$$Seq_i = \{C(s_0), C(s_1), \cdots, C(s_n)\}$$

with s_j being the ordered states within a session such that Seq_i is a continuous subsequence of that session. In the following sections, we explain the method used to derive the $Train$ and $Test$ datasets from our primary dataset $Data$.

The primary objective, naturally, of training a model of this kind is to utilize it for querying future user states, essentially engaging in predictive sequencing. This can be done by querying model M with the prefix of the anticipated user-state. In other words, since the model is trained on n-grams of user states, the (n-1)-gram preceding the anticipated action can be used to query for the forthcoming action. The $Encode$ function allows for this (n-1)-gram prefix to be represented within the n-gram's context as a query vector $q = P(Encode(Seq'_k[0:n-1]))$, where Seq'_k is intended to predict its n-th element.

By using our query vector q with our Model M through the binding operator $\otimes$, and due to the quasi orthogonality of the vectors, we solely unbind the suffix vector with a similar prefix (refer to ...). The resulting vector can be interpreted as a frequency vector representing the states linked to that prefix, expressed as $R = M * q = (C(s_0), C(s_1), \ldots, C(s_l))$, where l is determined by the occurrences of specific patterns within the training dataset. Because vector R is a frequency vector, performing a similarity analysis of R against all valid keys yields a discrete distribution pointing to the similarity with a specific state. A valid key k is characterized by $k \in \text{Domain}(C')$, implying that the collection of all valid keys corresponds to the items for which mapping is available in the function C', namely $\text{Domain}(C')$.

Thus, based on the frequency vector R, we can predict the n-th element in the sequence using $s_n = C'(\arg\max_k(Sim(R,k)))$. This methodology enables us to effectively perform predictive sequencing connected to action states.

5.2 Adaptiveness

Enhancing the foundation of the non-adaptive model, we propose altering our model's definition to enable greater flexibility. To accomplish this, we must revise the structure of our model M. This new iteration, model M, consists of two segments: M_{base}, akin to what was previously mentioned. Additionally, our model now incorporates an adaptive section, M_{adap}, which mirrors the base model's design. The complete model M is formed by combining these two segments using the bundling $\oplus$ operator. The inclusion of this adaptive component supports online learning and facilitates incremental training, much like the base model. In the following section, we will examine how this adaptation influences model performance and evaluate the computational resources demanded by this additional component. The implementation details of our methodology, along with the evaluation procedure and results, are accessible in the paper's replication package[6].

6 Evaluation Methodology

6.1 Datasets

Due to the challenges in accessing actual user data, we decided to utilize an already available dataset from a prior study, as outlined in the background section. Mozannar et al. [15] developed a taxonomy comprising twelve user states and gathered data from 21 developers to compile a labeled dataset of user states. This dataset can be employed to train and test the model in a constrained environment, allowing future opportunities for more comprehensive evaluation. Our preliminary analysis reveals that three labels provide limited information, leading us to exclude them from this study. Consequently, we are left with a total of 9 labels.

6.2 Data Partitioning Strategies

Let $Data$ denote the complete dataset of user session state sequences, where $Data = \{Sess_{u,i} \mid u \in U', i \in \{1, \ldots, n_u\}\}$, $U' \subset U$, and n_u is the number of sessions for user u. The task is to split $Data$ into two disjoint subsets, $Train$ and $Test$, such that $Train \cup Test = Data$ and $Train \cap Test = \emptyset$. We define three data partitioning strategies as follows.

6.2.1 Disjoint Split. In a disjoint split, the dataset is divided by users: one group for training and another for testing. We use data from 18 developers for training and the rest for testing.

6.2.2 Overlapping Split. In the overlapping split, each user's data is divided between training and testing sets. For example, with 10 sessions, 8 are for training and 2 for testing.

6.2.3 K-Fold Cross-Validation. This strategy creates a fold for each user by excluding them from training and using them only for evaluation, repeating for every user in the dataset.

6.3 Hyper-Parameters

In addition to exploring various data-splitting strategies, we investigate the impact of four other model (hyper-)parameters: dimension, subsequence length, cyclic shift, and adaptiveness. The dimension D was adjusted, selecting from fixed values $\{1000, 5000, 10000, 20000\}$. Similarly, the sequence length was altered using the set $\{3, 5, 7, 9\}$. Moreover, within the MAP framework, where the P operator acts as a cyclic shift of the representation vector, we assess the effect by altering the shift values among $\{2, 4, 6\}$. Lastly, we compare two proposed models: one incorporating the adaptive model and another without it.

6.4 metrics

6.4.1 Overall Accuracy. According to our setup, we assess the model's overall accuracy using the test set. When implementing

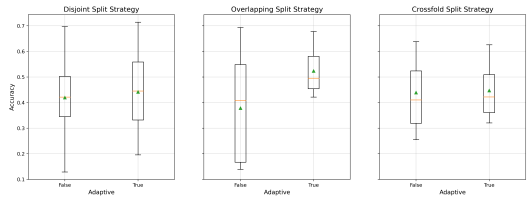


Figure 1: Overall accuracies of the model across the different splitting strategies and with adaptive on or off.

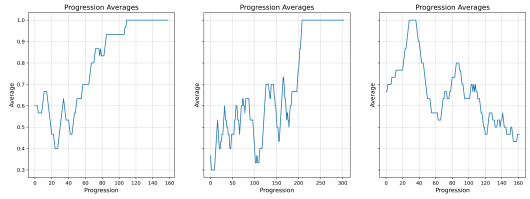


Figure 2: Best-performing model's accuracy; Sliding window size = 30, Dimension = 20,000, Sequence Length = 3, Cyclic Shift = 4, Adaptive = True, and Split Strategy = Disjoint.

cross-validation with several test sets, we calculate the average accuracy. For all other scenarios, we determine the model's accuracy considering every available user.

6.4.2 Sliding Window Accuracy. For adaptive models, we assess the sliding window accuracy to understand how the model's performance evolves. Specifically, a window of size k is used to traverse the prediction sequence, and the model's accuracy is computed within this window. This approach helps us determine the effectiveness of employing online learning to tailor models to users.

7 Preliminary Results

7.1 Model Performance

Our findings indicate that the models, on average, can perform within a range of 40 to 50 percent, with some configurations reaching above 70% on average. Furthermore, as depicted in Figure 1, the adaptive version of the model consistently surpasses the performance of the base version without employing any online learning.

Our assessments indicate that the cyclic shift parameter has a negligible impact on performance. Interestingly, our analysis of the results demonstrate that longer subsequences do not enhance predictive capabilities. In fact, an increase in sequence length corresponds with a decline in accuracy. This may result from the vector's information capacity being surpassed, suggesting further research is needed to determine if this pattern persists in higher dimensions.

Moreover, when examining the top-performing model, we observe a noteworthy trend in two of the three test scenarios: there is an upward trajectory in the model's average accuracy. This pattern suggests that online learning effectively enhances the model's performance. But more interestingly, there is not only an upward trend but rather that adapting the model can achieve results that make the predictions of the model near perfect.

7.2 Model Efficiency

7.2.1 Training Efficiency. Training the model involves a single pass through the training data. N-grams can be constructed in constant time, provided that previous sequences within a session are available and that the cold start of the session is amortized (see the appendix). Since the model operates on D-dimensional vectors, the time complexity for training is $O(D \cdot |Train|)$.

7.2.2 Update Efficiency. The update logic is implemented as a constant amount of operations on D dimensional vector, making the update $O(D)$.

7.2.3 Storage Efficiency. The storage requirements for the model depend on several components, which vary based on the specific implementation of the approach. To optimize memory usage, the domain of stored values per entry can be limited, allowing for storage in fewer bits than the conventional 32 or 64 bits. For instance, using 8 or 16 bits per entry can enhance storage efficiency. Let B represent the number of bits per entry. The stored items include the model memory (doubled in the case of the adaptive model) and the codebook. Consequently, the space complexity of the model is $O(B \cdot (|C| + D))$.

8 Future Work

In future work, we intend to broaden our approach for more extensive, real-world applicability. We plan to enhance the proposed framework to process incoming telemetry data from users similarly to other current methodologies. Moreover, we seek to leverage the predicted user state to adaptively optimize the user experience within the IDE. Furthermore, we are interested in investigating the impact of different Vector-Symbolic Architectures and assessing how they influence model accuracy.

9 Conclusion

This paper introduced HyperSeq, a Hyper-Adaptive Representation for Predictive Sequencing of States, as a novel, efficient approach to predicting developer actions in IDEs. Using hyperdimensional computing and online adaptation, HyperSeq achieved competitive accuracy, surpassing 70% in certain configurations, while maintaining low computational and storage requirements. Our evaluation demonstrated that HyperSeq not only excels in forecasting user actions but also dynamically improves its accuracy through online learning, enabling personalized predictions for individual users. This efficiency positions HyperSeq as a promising solution to reduce computational overhead in AI-powered IDEs and enhance user experiences through context-aware proactive adaptations. Future work will focus on integrating telemetry-based predictions, exploring alternative vector-symbolic architectures, and deploying HyperSeq in real-world environments to optimize developer productivity and promote sustainable computing practices.

References

[1] 2024. Llama 3.1 - 405B, 70B and 8B with multilinguality and long context. https://huggingface.co/blog/llama31
[2] 2024. U.S. Energy Information Administration (EIA). https://www.eia.gov/tools/faqs/faq.php?id=74&t=11
[3] 2025. Cursor - the AI code editor. https://www.cursor.com/
[4] 2025. GitHub Copilot · Your AI pair programmer. https://github.com/features/copilot

[5] 2025. JetBrains AI service and In-IDE AI Assistant. https://www.jetbrains.com/ai/
[6] Anonymous Authors. 2025. Replication Package. https://github.com/hyperseq-replication/package
[7] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2022. Grounded Copilot: How Programmers Interact with Code-Generating Models. arXiv:2206.15000 [cs.HC] https://arxiv.org/abs/2206.15000
[8] Aral de Moor, Arie van Deursen, and Maliheh Izadi. 2024. A transformer-based approach for smart invocation of automatic code completion. In *Proceedings of the 1st ACM International Conference on AI-Powered Software*. 28–37.
[9] Ross W Gayler. 1998. Multiplicative binding, representation operators & analogy (workshop poster). (1998).
[10] Ross W Gayler. 2004. Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. *arXiv preprint cs/0412059* (2004).
[11] Aaron Grattafiori and Abhimanyu Dubey et. al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783
[12] Michael Hersche, Mustafa Zeqiri, Luca Benini, Abu Sebastian, and Abbas Rahimi. 2023. A Neuro-vector-symbolic Architecture for Solving Raven's Progressive Matrices. arXiv:2203.04571 [cs.LG] https://arxiv.org/abs/2203.04571
[13] Pentti Kanerva. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation* 1 (2009), 139–159.
[14] Spencer J. Kent, E. Paxon Frady, Friedrich T. Sommer, and Bruno A. Olshausen. 2020. Resonator Networks outperform optimization methods at solving high-dimensional vector factorization. arXiv:1906.11684 [cs.NE] https://arxiv.org/abs/1906.11684
[15] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2024. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. arXiv:2210.14306 [cs.SE] https://arxiv.org/abs/2210.14306
[16] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2024. When to show a suggestion? Integrating human feedback in AI-assisted programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 10137–10144.
[17] Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. 2024. Leave No Context Behind: Efficient Infinite Context Transformers with Infini-attention. arXiv:2404.07143 [cs.CL] https://arxiv.org/abs/2404.07143
[18] Tony A Plate. 1995. Holographic reduced representations. *IEEE Transactions on Neural networks* 6, 3 (1995), 623–641.
[19] Agnia Sergeyuk, Ekaterina Koshchenko, Ilya Zakharov, Timofey Bryksin, and Maliheh Izadi. 2024. The Design Space of in-IDE Human-AI Experience. arXiv:2410.08676 [cs.SE] https://arxiv.org/abs/2410.08676
[20] Thomas Weber, Rafael Vinicius Mourao Thiel, and Sven Mayer. 2023. Supporting Software Developers Through a Gaze-Based Adaptive IDE. In *Proceedings of Mensch Und Computer 2023* (Rapperswil, Switzerland) *(MuC '23)*. 267–276. https://doi.org/10.1145/3603555.3603571