Quaternions are a useful tool in many areas of mathematics and physics, and the paper discusses their application in predictive analytics and feature interaction modeling. Specifically, it introduces a new approach called Quaternion Factorization Machines (QFM) that leverages quaternions to model complex feature interactions efficiently. The QFM approach is compared with traditional factorization machines and neural factorization machines, showing improved performance in terms of parameter size and prediction accuracy.

The paper's main contributions include:

- Introducing Quaternion Factorization Machines (QFM) as a lightweight and memory-efficient solution for feature interaction modeling.
- Comparing QFM with state-of-the-art methods, showing improvements in parameter size reduction and predictive accuracy.

The authors, Tong Chen, Hongzhi Yin, Xiangliang Zhang, Zi Huang, Yang Wang, and Meng Wang, discuss the motivation behind this work, which is to address the heavy parameterization of deep learning models, particularly in the context of predictive analytics and feature interactions. They argue that by using quaternions, they can model higher-order interactions efficiently, leading to better performance and smaller models.

The paper's introduction sets the stage for understanding the complex nature of feature interactions and the limitations of existing models. It motivates the need for new approaches that can handle the vast amount of data and complex relationships involved in modern predictive analytics tasks.

The main contributions are laid out in the body of the paper, with a detailed discussion of the Quaternion Factorization Machines and their performance evaluation. The authors provide extensive experimental results that demonstrate the effectiveness of their approach compared to existing methods.

The conclusion of the paper highlights the potential of quaternion-based models in predictive analytics, particularly in resource-constrained environments like IoT and edge devices. The authors also discuss the broader implications of their work, suggesting avenues for future research in quaternion-based modeling and its applications.
factorization machine (NFM). Instead of the straightforward inner product in FM, NFM takes the sum of all features’ linear pairwise combinations into a feed-forward neural network, and generates a latent representation of high-order feature interactions. With the idea of learning high-order feature interactions with DNNs, various DNN-based approaches are devised for predictive analytics [8], [19]–[23] and demonstrate advantageous prediction performance.

However, compared with the plain FM, all the aforementioned DNN-based variants are heavily parameterized and thus memory-inefficient. Consequently, the achieved performance gain comes at a high cost of introducing excessive parameters in their neural architectures. For instance, the optimal performance of xDeepFM [21] is commonly achieved with a configuration of at least three convolution layers joined by two feed-forward deep layers [15], [21], [24], leading to millions of additional model parameters on top of the plain FM (see Section VII). As a result, the bulkiness of existing deep FM-based models inevitably renders practical and efficient model deployment challenging. These memory-intensive models are also too cumbersome to welcome the emerging opportunities of applications on mobile or IoT devices [25]. Furthermore, as another side effect, the excessive parameterization creates an obstacle for effectively learning useful interaction patterns from a large number of parameters, especially with sparse features [24]. As more sophisticated DNN structures are being utilized to model feature interactions (e.g., the graph neural networks in [26] and transformers in [15]), enhancing FM with DNNs appears to be an ongoing trend. In light of this, it is crucial to derive a new paradigm that only requires lightweight parameterization just like the plain FM, while allowing for comprehensive feature interaction modelling to generate accurate predictions.

Before taking the step to derive a corresponding solution, a key observation we have drawn in this line of research is that most work has primarily focused on real-valued representations in the $\mathbb{R}$ space. In contrast, building machine learning models with the notion of quaternion algebra [27] in the hypercomplex space $\mathbb{H}$ has just started revealing its immense potential through recent advances [28]. In quaternion algebra, the Hamilton product plays a key role in supporting intricate interactions between quaternion representations while minimizing the parameter consumption. Compared with the matrix multiplication that is common in all DNN-based models’ deep layers [8], [14], [21], [24], the Hamilton product between two quaternions has fewer degrees of freedom, enabling up to four times compression of parameter size in a single multiplication operation. Moreover, quaternions are hypercomplex numbers consisting of one real and three imaginary components, in which the inter-dependencies between these components are naturally encoded during training via the Hamilton product [29]. Thus, machine learning approaches that utilize quaternion-valued representations and operations are inherently advantageous in modelling the subtle yet complex interactions between features, reflected by an emerging surge in successful applications like image processing [30], [31], speech recognition [32], and text classification [29]. More recently, [33] successfully rolls out a quaternion collaborative filtering framework by exploiting the computation of quaternion algebra to quantify the affinity between users and items. Hence, this verifies the capability of quaternions in modelling subtle interactions among latent representations, which is also the essence of the plain FM and its successors.

To this end, to bypass the limitations of using real-valued DNNs for feature interaction modelling, we move beyond the real space, and subsume FM-based models under a quaternion-valued learning framework for the first time. In this paper, we propose the first quaternion factorization machine (QFM) that generalizes the original FM into the hypercomplex domain, and then further infuse non-linear expressiveness into QFM by extending it to a deep neural model, namely quaternion neural factorization machine (QNFm). By making full use of quaternions, we create an ideal bootstrap between a compact model size and uncompromised performance. In both models, we parameterize the feature interaction schemes as quaternion-valued functions in the hypercomplex space $\mathbb{H}$, and each feature embedding diverges into four quaternion components. In QFM, to thoroughly model feature interactions, we replace the shallow dot product in the plain FM with our proposed inner Hamilton product for quaternion components, in the hypercomplex space. In contrast, building machine learning models with the notion of quaternion algebra [27] in the hypercomplex space $\mathbb{H}$ has just started revealing its immense potential through recent advances [28]. In quaternion algebra, the Hamilton product plays a key role in supporting intricate interactions between quaternion representations while minimizing the parameter consumption. Compared with the matrix multiplication that is common in all DNN-based models’ deep layers [8], [14], [21], [24], the Hamilton product between two quaternions has fewer degrees of freedom, enabling up to four times compression of parameter size in a single multiplication operation. Moreover, quaternions are hypercomplex numbers consisting of one real and three imaginary components, in which the inter-dependencies between these components are naturally encoded during training via the Hamilton product [29]. Thus, machine learning approaches that utilize quaternion-valued representations and operations are inherently advantageous in modelling the subtle yet complex interactions between features, reflected by an emerging surge in successful applications like image processing [30], [31], speech recognition [32], and text classification [29]. More recently, [33] successfully rolls out a quaternion collaborative

This paper brings the following contributions:

- We address a common drawback of existing DNN-based FM variants, i.e., the heavy parameterization inherited from real-valued neural networks. We sought solutions from the hypercomplex space by exploiting quaternion representations. To the best of our knowledge, this is the first exploration of quaternion FM-based models.
- We propose QFM and QNFm, two novel and lightweight methods for predictive analytics. By utilizing quaternion representations and defining interaction paradigms with the notion of Hamilton product, our models are highly expressive when modelling the intricate feature interactions, and can greatly reduce the parameter consumption.
- We extensively evaluate our models on three large-scale real datasets. Experiments show that QFM yields considerable improvements over the plain FM with no extra parameter costs, while QNFm achieves the state-of-the-art prediction accuracy with a parameter reduction by up to two magnitudes regarding its real-valued counterparts.
A. Notations

Throughout this paper, all vectors and matrices are respectively denoted by bold lower case and bold upper case letters, e.g., \( \mathbf{x} \) and \( \mathbf{X} \), and all vectors are column vectors unless specified. To maintain the simplicity, we use superscript \( \diamond \) to distinguish all quaternion-valued parameters. Sets are represented by calligraphic uppercase letters, e.g., \( \mathcal{V} \). Hallowed uppercase letters \( \mathbb{H} \) and \( \mathbb{R} \) are respectively sets of quaternions and real numbers, while we use \( \mathbb{I}, \mathbb{J}, \mathbb{K} \) to denote three imaginary numbers composing a quaternion.

B. Factorization Machines

Factorization machines (FMs) [16] are originally proposed for collaborative recommendation. Take instance \([\text{user ID} = 2, \text{gender} = \text{female}, \text{movie} = \text{Avengers, movie type} = \text{action}]\) as an example, its input is a high-dimensional sparse feature \( \mathbf{x} \in \{0, 1\}^n \) constructed by concatenating the one-hot encodings from all feature fields [8], [21], [23]:

\[
\mathbf{x} = \begin{bmatrix} 0, 1, 0, \ldots, 0 \end{bmatrix} \begin{bmatrix} 1, 0 \end{bmatrix} \begin{bmatrix} 0, 1, 0, \ldots, 0 \end{bmatrix} \begin{bmatrix} 0, 1, 0, \ldots, 0 \end{bmatrix},
\]

where any real-valued feature (e.g., age) can also be directly included in \( \mathbf{x} \). Then, FMs are linear predictors that estimate the desired output by modelling all interactions between each pair of features within \( \mathbf{x} \):

\[
\hat{y} = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} v_i^T v_j \cdot x_i x_j,
\]

where \( n \) is the total number of features, \( w_0 \) is the global bias, \( w_i \) is the weight assigned to the \( i \)-th feature, and \( x_i, x_j \in \mathbf{x} \) are respectively the observed values of the \( i \)-th and \( j \)-th features. \( v_i, v_j \in \mathbb{R}^d \) are corresponding embedding vectors for features \( i \) and \( j \), while \( d \) is the embedding dimension. Thus, the first two terms in Eq.(2) can be viewed as a linear regression scheme, while the third term models the effect of pairwise feature interactions [19].

C. Quaternion Algebra

Quaternion is an extension of complex number that operates between each pair of features within \( \mathbf{x} \):

\[
\hat{y} = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} v_i^T v_j \cdot x_i x_j,
\]

where \( n \) is the total number of features, \( w_0 \) is the global bias, \( w_i \) is the weight assigned to the \( i \)-th feature, and \( x_i, x_j \in \mathbf{x} \) are respectively the observed values of the \( i \)-th and \( j \)-th features. \( v_i, v_j \in \mathbb{R}^d \) are corresponding embedding vectors for features \( i \) and \( j \), while \( d \) is the embedding dimension. Thus, the first two terms in Eq.(2) can be viewed as a linear regression scheme, while the third term models the effect of pairwise feature interactions [19].

\[ q^0 = r + aI + bJ + cK, \]

where \( r, a, b, c \in \mathbb{R} \) are real numbers, while \( 1, I, J \) and \( K \) are the quaternion unit basis. In the Hamilton quaternion space \( \mathbb{H} \), a quaternion \( q^0 \) consists of one real part and three imaginary parts:

\[ q^0 = r + aI + bJ + cK, \]

where \( r, a, b, c \in \mathbb{R} \) are real numbers, while \( 1, I, J \) and \( K \) are the quaternion unit basis. In the rest of our paper, we term the four components \( r, a, b, c \) as quaternion cores and simplify the real part \( rI1 \) as \( r \) to be concise. \( I, J \) and \( K \) are imaginary numbers and satisfy the following Hamilton’s rule:

\[ I^2 = J^2 = K^2 = IJK = -1, \]

from which multiple useful properties can be derived, e.g., \( I^2JK = -J \) leads to \( JJK = I \). To list a few:

\[ IJ = K, JI = -K, KJ = I, KI = J, IK = -J. \]

In \( \mathbb{H} \), a normalized quaternion, i.e., unit quaternion \( q^0 \) is expressed as:

\[ q^0 = \frac{q}{|q^0|} = \frac{q}{\sqrt{r^2 + a^2 + b^2 + c^2}}. \]

The multiplication, i.e., Hamilton product of two quaternions \( q_1 = r_1 + a_1I + b_1J + c_1K \) and \( q_2 = r_2 + a_2I + b_2J + c_2K \) can be computed following the distributive law:

\[ q_1^0 \times q_2^0 = (r_1r_2 - a_1a_2 - b_1b_2 - c_1c_2)I + (r_1a_2 + a_1r_2 + b_1c_2 - c_1b_2)J + (r_1b_2 - a_1c_2 + b_1r_2 + c_1a_2)K + (r_1c_2 + a_1b_2 - b_1a_2 + c_1r_2)J, \]

where we should note that based on the rules in Eq.(4) and Eq.(5), the Hamilton product is associative but non-commutative, i.e., \( q_1^0 \times q_2^0 \neq q_2^0 \times q_1^0 \).

III. Quaternion Factorization Machines

To start with, we firstly introduce the basic form of QFM by partially transferring FM to the quaternion space. In QFM, apart from the real-valued sparse input \( \mathbf{x} \) and output \( \hat{y} \), we retain the first two regression terms in Eq.(2) in the real space, while all model parameters and intermediate computations for the feature interactions are defined in the quaternion space. Mathematically, QFM is written as:

\[ \hat{y} = w_0 + \sum_{i=1}^{n} w_i x_i + \phi(\mathbf{V}^0), \]

where \( w_0, w_i \in \mathbb{R} \) are real-valued global bias and weight associated with the \( i \)-th feature. \( \mathbf{V}^0 \) is the set of quaternion embedding vectors for all non-zero features in \( \mathbf{x} \), \( \phi(\cdot) \) is a quaternion-valued function that firstly quantifies the intensity of all pairwise feature interactions in the \( \mathbb{H} \) space, and maps the quaternion results into real numbers. In what follows, we introduce the design of \( \phi(\cdot) \) in detail.

A. Quaternion Embeddings

In FM-based methods, the real-valued features are converted into \( d \)-dimensional vector representations (a.k.a. embedding vectors) to allow for the modelling of pairwise feature interactions. In the quaternion space, we embed features with quaternion vectors. Based on the construction rule of \( \mathbf{x} \in \mathbb{R}^n \), it can be viewed as the additive form of the one-hot encodings for all non-zero features. Thus, \( \mathbf{x} = \sum_{i=1}^{n} x_i \mathbf{e}_i \), where \( \mathbf{e}_i = [0, ..., 0, 1, 0, ..., 0] \) is an \( n \)-dimensional one-hot vector of an individual non-zero feature. Specifically, we define a quaternion embedding vector \( \mathbf{V}^0 \in \mathbb{H}^d \) as \( \mathbf{V}^0 = r + aI + bJ + cK \), where \( r, a, b, c \in \mathbb{R}^d \) are real-valued vectors. Then, our quaternion embedding system consists of four distinct
embedding matrices to match up with four quaternion cores in \( v_i^0 \) for the \( i \)-th feature:

\[
v_i^0 = r_i + a_i \mathbb{I} + b_i \mathbb{J} + c_i \mathbb{K}
\]

(9)

where \( M_r, M_a, M_b, M_c \in \mathbb{R}^{n \times d} \) are four corresponding embedding matrices, while each embedding component is weighted by the scalar value of feature \( x_i \). Due to the sparse nature of \( x \), we only need to include the embeddings of non-zero features in \( V_x^0 \), i.e., \( V_x^0 = \{ v_i^0 | x_i \neq 0 \} \) for subsequent computations.

### B. Feature Interaction Modelling with Inner Hamilton Product

With all quaternion feature embeddings \( V_x^0 \), we propose a pairwise feature interaction scheme with an inner Hamilton product operation that we have defined below to enable cross-features in the quaternion space:

\[
h^0 = \sum_{i=1}^{n} \sum_{j=1}^{n} v_i^0 \otimes v_j^0
\]

(10)

\[
h^0 = r_h + a_h \mathbb{I} + b_h \mathbb{J} + c_h \mathbb{K} \in \mathbb{H}
\]

is a quaternion that encodes all the pairwise interaction information among features in \( V_x^0 \). \( \otimes \) denotes our proposed inner Hamilton product of two quaternion vectors. For an arbitrary pair of quaternion embeddings \( v_i^0, v_j^0 \in \mathbb{H}^d \), \( v_i^0 \otimes v_j^0 \) is computed as follows:

\[
v_i^0 \otimes v_j^0 = (r_i^0 r_j - a_i^0 a_j - b_i^0 b_j - c_i^0 c_j) + (r_i^0 a_j + a_i^0 r_j + b_i^0 c_j - c_i^0 b_j) \mathbb{I}
\]

+ \( (r_i^0 b_j - a_i^0 c_j + b_i^0 r_j + c_i^0 a_j) \mathbb{J} \)

+ \( (r_i^0 c_j + a_i^0 b_j - b_i^0 a_j + c_i^0 r_j) \mathbb{K} \)

(11)

where the two quaternion cores interact with each other via inner product, thus yielding a quaternion scalar as the output. It is worth noting that unlike the traditional pairwise feature interaction scheme in real-valued FMs (i.e., \( v_i^0 v_j \) in Eq.(2)), we allow the \( i \)-th and \( j \)-th features to interact with each other twice, i.e., \( v_i^0 \otimes v_j^0 \) and \( v_j^0 \otimes v_i^0 \). This is because our proposed inner Hamilton product inherits the non-commutative nature from the Hamilton product, making \( v_i^0 \otimes v_j^0 \) and \( v_j^0 \otimes v_i^0 \) non-interchangeable and carry asymmetric interaction contexts. Correspondingly, instead of the straightforward vector inner product, Eq.(10) allows for effective infusion of bilateral pairwise feature interactions, thus providing richer predictive power in return.

### C. Rationale of Hamilton Product in Feature Interactions

The quaternion representation \( v_i^0 \in \mathbb{H}^d \) of feature \( i \) can be thought of as a combination of four information channels, i.e., \( 1, \mathbb{I}, \mathbb{J}, \mathbb{K} \). Owing to the fact that \( v_i^0 \in \mathbb{H}^d \) carries four distinct \( d \)-dimensional real vectors, it can be viewed as evenly dividing a \( 4d \)-dimensional vector \( [r_i; a_i; b_i; c_i] \) into a 4-channel format. Each information channel carries a real-valued, \( d \)-dimensional dense vector \( r_i, a_i, b_i, \) and \( c_i \). Hence, the inner Hamilton product \( \otimes \) between two quaternion embeddings creates a cross-channel interaction scheme following Hamilton’s rule. As illustrated in Figure 1, when computing \( r_i + a_i \mathbb{I} + b_i \mathbb{J} + c_i \mathbb{K} = v_i^0 \otimes v_j^0 \), the interaction response in each channel is generated by applying different combinatorial rules to the dot products of real-valued cores in \( v_i^0 \) and \( v_j^0 \). Intuitively, the quaternion scalar resulted from inner Hamilton product compresses the signal of cross-channel information flows from one feature to the other, representing their pairwise interaction intensity.

### D. Generating Predictions

In \( \phi() \), the last step is to map the quaternion output \( h^0 \) back to the real space, so that the learned feature interaction information can be coupled to the real-valued linear regression term to generate accurate predictions. Since each core in \( h^0 \) can be viewed as an interaction score learned in a distinct channel, we adopt average pooling to merge the information from all channels:

\[
\phi(V_x^0) = f_{\mathbb{H} \rightarrow \mathbb{R}}(h^0) = \frac{r_h + a_h + b_h + c_h}{4},
\]

(12)

which will be used in Eq.(8) to emit the final prediction \( \hat{y} \).

### IV. Extending QFM: Quaternion Neural Factorization Machines

To harness the advantages of nonlinearity in feature interaction modelling and strengthen the expressiveness of QFM, we further propose quaternion neural factorization machine (QNFM), a neural extension of QFM. Specifically, in QNFM, we preserve the first two linear regression terms in Eq.(8), and infuse nonlinearity by replacing the linear interaction function...
\( \phi(\cdot) \) with a quaternion neural network-based architecture \( \phi'(\cdot) \). The output of QNFM is then \( \hat{y} = w_0 + \sum_{i=1}^{n} w_i x_i + \phi'(V_x^\omega) \). In this section, we present our pathway to building the function \( \phi'(\cdot) \) in QNFM.

### A. Asymmetrical Quaternion Interaction Pooling

Similar to FM-based models built upon real-valued deep neural networks, all quaternion feature embeddings in QNFM need to be combined into one unified representation before they are fed into the neural networks to model the complex and non-linear feature interactions. Common approaches include concatenations [8], [21] and pooling operations [14], [19] for all feature embeddings. As concatenating all feature embeddings will produce a high-dimensional vector, it will lead to the huge dimensionality of corresponding network parameters (i.e., weights and biases) and increased model sizes. Also, since \( V_x^\omega \) only contains the embeddings of non-zero features, the dimensions of resulted concatenations may vary for different feature embedding sets \( V_x^\omega \) in some cases, making the subsequent neural network ill-posed. On this occasion, pooling operation is a better fit for QNFM as it aims to compress all feature embeddings into a compact and fix-size vector. Thus, we propose \textit{asymmetrical quaternion interaction pooling} in QNFM to merge all embeddings in \( V_x^\omega \) into a condensed quaternion-valued representation \( \tilde{\mathbf{v}}^\omega \in \mathbb{H}^d \):

\[
\tilde{\mathbf{v}}^\omega = \sum_{i=1}^{n} \sum_{j=1}^{n} (\mathbf{v}_i^\omega \odot \mathbf{v}_j^\omega),
\]

(13)

where \( \odot \) denotes the \textit{element-wise Hamilton product} we define for two \( d \)-dimensional quaternion vectors. Take \( \mathbf{v}_i^\omega \odot \mathbf{v}_j^\omega \) as an example, then:

\[
\begin{align*}
\mathbf{v}_i^\omega \odot \mathbf{v}_j^\omega &= (r_i r_j - a_i a_j - b_i b_j - c_i c_j) \\
&+ (r_i a_j + a_i r_j + b_i c_j - c_i b_j) \mathbf{I} \\
&+ (r_i b_j - a_i c_j + b_i r_j + c_i a_j) \mathbf{J} \\
&+ (r_i c_j + a_i b_j - b_i a_j + c_i r_j) \mathbf{K},
\end{align*}
\]

(14)

where \( \odot \) represents the element-wise multiplication between two real-valued vectors. Similar to the inner Hamilton product, the element-wise Hamilton product \( \odot \) also allows for the cross-channel information exchange between two quaternion embedding vectors as shown in Figure 1, where the quaternion cores interact with each other through element-wise product rather than inner product. For \( \mathbf{v}_i^\omega \) and \( \mathbf{v}_j^\omega \), the result of their element-wise Hamilton product is still a \( d \)-dimensional quaternion vector. Thus, compared with inner Hamilton product, element-wise Hamilton product has a lower compression rate when encoding pairwise feature interactions, but it preserves substantially more interaction contexts and makes Eq.(13) an ideal pooling operation for the subsequent quaternion-valued neural network to learn expressive latent representations.

### B. Quaternion Feed-Forward Network

With the quaternion asymmetrical interaction pooling, all pairwise feature interactions are summarized in a unified representation \( \tilde{\mathbf{v}}^\omega \). However, it is still a linear computation process. To further model the complex and non-linear interactions between different latent dimensions, we stack a quaternion-valued \( l \)-layer residual feed-forward network upon the interaction pooling layer:

\[
\begin{align*}
\tilde{h}_0^\omega &= \tilde{\mathbf{v}}^\omega + \sigma(\mathbf{W}_1^r \times \tilde{\mathbf{v}}^\omega + \mathbf{b}_1^\omega), \\
\tilde{h}_2^\omega &= \tilde{h}_1^\omega + \sigma(\mathbf{W}_2^r \times \tilde{h}_1^\omega + \mathbf{b}_2^\omega), \\
&\vdots \\
\tilde{h}_n^\omega &= \tilde{h}_{n-1}^\omega + \sigma(\mathbf{W}_n^r \times \tilde{h}_{n-1}^\omega + \mathbf{b}_n^\omega),
\end{align*}
\]

(15)

where \( \mathbf{W}_n^r = \mathbf{R}_{w_i} + \mathbf{A}_{w_i} \mathbf{I} + \mathbf{B}_{w_i} \mathbf{J} + \mathbf{C}_{w_i} \mathbf{K} \in \mathbb{H}^{d \times d} \) is the quaternion weight matrix, \( \mathbf{h}_n^\omega, \mathbf{b}_n^\omega \in \mathbb{H}^d \) are respectively the latent representation and bias in each layer, while \( \sigma(\cdot) \) is the non-linear activation function. In the following, we introduce the three key components in the quaternion feed-forward network.

#### Split Activation Function.

Activation functions plays a crucial role in modelling nonlinearity. Though there are few attempts to devise fully quaternion-valued activation functions by extending real-valued functions (e.g., tanh function) to the hypercomplex domain [35], [36], the quaternion-valued activation functions need a careful training phase due to an important number of singularities that can drastically alter a network’s performance [28], [32]. As such, we adopt split activation for quaternion-valued neurons, which is widely adopted in [29], [31], [32] due to its higher stability and simpler computation process. Specifically, for layer \( l' \leq l \), let \( \mathbf{h}_l^\omega = r_h + a_h \mathbf{I} + b_h \mathbf{J} + c_h \mathbf{K} = \mathbf{W}_l^r \times \tilde{h}_{l-1}^\omega + \mathbf{b}_l^\omega \), then \( \sigma(\mathbf{h}_l^\omega) \) is defined as:

\[
\sigma(\mathbf{h}_l^\omega) = \sigma(r_h + a_h \mathbf{I} + b_h \mathbf{J} + c_h \mathbf{K})
\]

(16)

\[
= \text{ReLU}(r_h) + \text{ReLU}(a_h) \mathbf{I} + \text{ReLU}(b_h) \mathbf{J} + \text{ReLU}(c_h) \mathbf{K},
\]

where \( \text{ReLU}(\cdot) \) is the rectified linear unit we have chosen for non-linear activation. With the split activation, the quaternion feed-forward network is able to simultaneously learn the latent representation of feature interactions through four separate channels (i.e., \( 1, \mathbf{I}, \mathbf{J}, \mathbf{K} \)), making the learned \( \mathbf{h}_n^\omega \) carry more intricate semantics about feature interactions.

#### Residual Connections.

The core idea behind residual networks is to propagate low-layer features to higher layers by residual connection [37]. By combining low-layer interaction features with the high-layer representations computed by the feed-forward network, the residual connections essentially allow the model to easily propagate low-layer features to the final layer, which can help the model enhance its expressiveness using different information learned hierarchically. Intuitively, in our shared quaternion feed-forward network, to generate a comprehensive representation for feature interactions in each view, the \( l' \)-th layer iteratively fine-tunes the representation learned by the \( (l' - 1) \)-th layer (i.e., \( \mathbf{h}_{l'-1}^\omega \)) by adding a learned residual, which corresponds to the second term in Eq.(15).

#### Layer Dropout.

To prevent QNFM from overfitting the training data, we adopt dropout [38] on all the layers of the quaternion feed-forward network as a regularization strategy. In short, we randomly drop the neurons with the ratio of \( \rho \in (0, 1) \) during training. Hence, dropout can be viewed as a form of ensemble learning which includes numerous models.
that share parameters [39]. It is worth mentioning that all the neurons are used when testing, which can be seen as a model averaging operation [38] in ensemble learning.

C. Comprehending Quaternion Feed-Forward Network

In Eq.(15), the standard form of Hamilton product is used between the quaternion weight matrix and the quaternion vector. Recall that we can view the pooled quaternion vector $\bar{v}$ as a 4-channel representation containing four quaternion cores. As discussed in [30], quaternion-valued matrix-vector multiplications is essentially mapping the original quaternion vector input to a new one by collectively using four sets of weights from the 1, I, J, K channels, as illustrated in Figure 1. Therefore, in the $l'$-th layer, $W^\circ_l \times h^\circ_{l-1}$ produces a $d$-dimensional vector, where every single channel consists of a unique linear combination of each information channel in $W^\circ_l$ and $h^\circ_{l-1}$. This comes from the properties of the Hamilton product, and is forcing each channel of the weight to interact with each channel of the input. Coupled with the bias terms, the split activation function and the residual connections, the quaternion feed-forward network is able to learn discriminative representations of subtle feature interactions via repetitive information exchange among the 1, I, J, K channels.

D. Output Layer

The final vector representation $h^\circ_L$ is utilized to compute the quaternion output for the factorized feature interaction component via the inner Hamilton product of the quaternion output for the factorized feature interaction $\tilde{y}$ and weight $p^\circ \in \mathbb{H}^d$:

$$\phi'(V^\circ_y) = f_{\mathbb{H}^d}(p^\circ \otimes h^\circ_L),$$

where $f_{\mathbb{H}^d}(\cdot)$ is the same mapping function as defined in Eq.(12). Lastly, we summarize the entire prediction result of QNFM as:

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i x_i + f_{\mathbb{H}^d}(p^\circ \otimes h^\circ_L).$$

V. MODEL LEARNING AND COMPLEXITY ANALYSIS

A. Model Learning

Loss Function. As the scopes of both the input and output are not restricted, QFM and QNFM are flexible and versatile models which can be adopted for different tasks. In this paper, we focus on click-through rate (CTR) prediction, which is one of the most popular applications for FM-based models in recent years [4], [8], [20]–[22]. Given an arbitrary user and features like her/his profile or previously visited links (e.g., web pages or advertisements), CTR prediction aims to predict the possibility of clicking through a given link. To optimize QFM and QNFM towards the task goal, we enforce the model to output a probability of observing a (user, link) instance. By replacing (user, link) with the notion ($u_i$, $v_j$), we quantify the prediction error with log loss, which is a special case of the cross-entropy:

$$L = - \sum_{(u_i, v_j) \in S} \left( y_{ij} \log \beta(\hat{y}_{ij}) + (1-y_{ij}) \log(1-\beta(\hat{y}_{ij})) \right),$$

where $\beta(\cdot)$ is the sigmoid function to enable our model’s classification capability such that $\beta(\tilde{y}_{ij}) \in (0, 1)$ represents the possibility that user $u_i$ will click through link $v_j$. $S$ is the set of ground truth ($u_i, v_j$) pairs labelled by $y_{ij} \in \{0, 1\}$.

Optimization Strategy. With the split activation in Eq.(16) and real-valued loss $L$, we can efficiently apply stochastic gradient descent (SGD) algorithms to learn the model parameters by minimizing $L$. Hence, we leverage a mini-batch SGD-based algorithm, namely Adam [40] optimizer. For different tasks, we tune the hyperparameters using grid search. Specifically, the latent dimension (i.e., factorization factor) $d$ is searched in $\{4, 8, 16, 32, 64\}$; the depth of the quaternion feed-forward network $l$ is searched in $\{1, 2, 3, 4, 5\}$; and the dropout ratio $\rho$ is searched in $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. We will further discuss the impact of these key hyperparameters to the prediction performance of QFM/QNFM in Section VII-G. In addition, we set the batch size to 512 according to device capacity and the learning rate to $1 \times 10^{-5}$. The early stopping strategy is adopted, where the training process is terminated if the log loss on validation data keeps increasing for three consecutive training epochs.

B. Space Complexity Analysis

Since a major benefit of utilizing quaternion representations in FM-based models is the avoidance of excessive parameters, we analyze the space complexity of QFM and QNFM in detail. First, as pointed out in Section III-C, a $d$-dimensional quaternion representation can be viewed as an unravelled 4d-dimensional real-valued vector. So, we should note that at the bit level, a $d$-dimensional quaternion vector has the same representation capacity as a vector in $\mathbb{R}^{4d}$, and will bring an identical number of 4d neurons in neural networks [29], [32]. In the same vein, our quaternion embedding scheme in Eq.(9) does not yield additional parameters as it just vertically splits a full embedding matrix in real-valued FM-models into 4 even partitions. Hence, we exclude the embedding layer in this discussion since it is mandatory in all FM-based models. Also, we should recall that all the learnable parameters in the plain FM (i.e., Eq.(2)) apart from feature embeddings are the global bias $w_0$ and feature weights $\{w_i\}_{i=1}^{n}$, bringing only $n+1$ parameters in total.

Space Complexity of QFM. In QFM, we keep the original linear regression terms as in FM and only make changes to the feature interaction scheme. As Eq.(8) suggests, the quaternion-valued feature interaction function does not introduce any additional parameters to learn. Thus, the number of trainable parameters in QFM is identical to the lightweight FM.

Space Complexity of QNFM. We start with the quaternion feed-forward network in QNFM, where its advantage of parameter reduction can be revealed in the comparison with real-valued deep networks. To be consistent, we benchmark the parameter size with 4d neurons in a neural network, which corresponds to $\mathbb{H}^d$ in the hypercomplex space as per our discussion. In each feed-forward layer of QNFM, the trainable parameters are the quaternion cores in weight $W^\circ$ and bias $b^\circ$, consuming $4d^2$ and $4d$ parameters per layer, respectively. In contrast, though an equivalent single-layer, real-valued neural
network with $4d$ neurons has a bias vector of the same size, its $4d \times 4d$ weight matrix yields $16d^2$ parameters in total. To explain where the lower parameter consumption comes from, we take $d = 1$ as an example in the $l'$-layer of Eq.(15). Intuitively, if we set aside the quaternion unit basis $1$, $\mathbb{I}$, $\mathbb{J}$ and $\mathbb{K}$, the Hamilton product $W_{l'}^h \times h_{l'}^w$ can be approximated as:

$$W_{l'}^h \times h_{l'}^w \sim \begin{bmatrix} r_w & -a_w & -b_w & -c_w \\ a_w & r_w & c_w & b_w \\ b_w & c_w & r_w & -a_w \\ c_w & -b_w & a_w & r_w \end{bmatrix} \times \begin{bmatrix} r_h \\ a_h \\ b_h \\ c_h \end{bmatrix}, \quad (20)$$

where there are only 4 distinct parameter variables (i.e., quaternion cores $r_w$, $a_w$, $b_w$ and $c_w$) in the weight matrix, leading to as few as 4 degrees of freedom in trainable parameters. However, in every real-valued feed-forward operation, all the 16 elements in the weight matrix are different parameter variables [29]. Hence, ignoring the small bias vector, our quaternion feed-forward operation achieves a $75\%$ reduction in parameterization. Meanwhile, similar to QFM, the size of model parameters in the first two regression terms of Eq.(15) is $n + 1$. In Eq.(13) which is our proposed asymmetrical quaternion interaction pooling layer, there are none learnable parameters. As the projection weight $p$ in the output layer has $4d$ parameters, the total parameter size of QNFM is $n + 1 + l \times (4d^2 + 4d) + 4d$. Considering the $n + 1$ parameters in the plain FM, the extra space complexity introduced by QNFM on top of FM is $O(d^2l + dl + d)$. On the contrary, an equivalently structured NFM [14] which is one of the simplest DNN-based FM variants, has the extra space complexity of $O(4d^2l + dl + d)$ versus FM. As $l \ll d$ in practice, our QNFM is a substantially more compact model than its real-valued counterparts, thus being highly memory-efficient.

C. Time Complexity Analysis

Excluding the embedding operation that is standard in all FM-based models, the major computational cost of our model is exerted by the feature interaction functions $\phi(\cdot)$ and $\phi'(\cdot)$ respectively in QFM and QNFM. In what follows, we prove that their time complexity is linearly associated to only the number of instances in a dataset.

**Time Complexity of QFM.** The core computational step in $\phi(\cdot)$ is the feature interaction scheme based on inner Hamilton product defined in Eq.(10), which has the time complexity of $O(n^2d)$ for each data instance. Because we only account for non-zero features (i.e., $\mathcal{V}_2$), the actual time complexity is $O(|\mathcal{V}_2|^2d)$ where $|\mathcal{V}_2| \ll n$. As the number of non-zero features (i.e., feature fields) in $\mathcal{V}_2$ is constant and commonly small in CTR prediction datasets (see Section VII-A), and the latent dimension $d$ is also fixed, QFM has linear time complexity w.r.t. the scale of the data.

**Time Complexity of QNFM.** In QNFM, the time complexity of $\phi'(\cdot)$ is mainly brought by the asymmetrical quaternion interaction pooling, quaternion feed-forward network, and the projection operation within the output layer, which respectively takes $O(|\mathcal{V}_2|^2d)$, $O(d^2)$ and $O(d)$ time to compute. Hence, the total time complexity tallies up to $O(|\mathcal{V}_2|^2d + d^2 + d)$ per instance, which is also linearly related to the data size.

VI. RELATED WORK

A. Feature Interaction-based Predictive Analytics

When performing sparse predictive analytics, learning the context of interactions among observed features is an indispensable means for machine learning models to achieve optimal results [8]. On this occasion, factorization machine (FM) [16] is a well-established model that captures the second-order feature interactions for predictive analytics. Following FM, a series of its linear variants are proposed, such as field-aware FM [4], higher-order FM [17] and importance-aware FM [41]. However, as stated in many literatures [2], [14], [21], [23], these FM extensions are constrained by their limited expressiveness when mining high-order interaction patterns and learning discriminative feature representations.

With the fruitful advances in deep neural networks (DNNs) [18], research that incorporates DNNs into the feature interaction paradigm has gained state-of-the-art performance in a considerable amount of sparse prediction tasks [14], [21], [23], [42]. Generally, there are two main themes of leveraging DNNs to learn feature interactions, which are developing “deep” and/or “wide” models [2], respectively. In a deep structure, multi-layer DNNs are commonly stacked upon linear components to exhaustively extract useful information from feature interactions, e.g., the residual network in DeepCrossing [8], the convolutional network in FGCNN [24], and the compressed interaction network in xDeepFM [6]. To build wide models, multiple types of feature interactions from varied domains (if available) are taken into account, e.g., separately modelling user logs and texts with CoFM [43], differentiating user and item contexts in ENSFM [44], and applying multi-view learning to dynamic and static features in SeqFM [2]. It is worth mentioning that, in recent hybrid models, by fusing shallow low-order (e.g., linear regression) output with dense high-order (e.g., DNN) output, deep and wide structures can be united to offer richer prediction signals, such as Wide&Deep [20], DeepFM [22], DCN [45], and AutoInt [15].

As pointed out in Section I, compared with the plain FM, though DNN-based methods are more advantageous in feature interaction modelling, the heavy parameterization is a mandatory prerequisite for those high-performance models. Consequently, the excessive parameter consumption of aforementioned deep models creates a severe bottleneck for efficient deployment and training in the real world.

B. Quaternion Representations and Neural Networks

In a nutshell, DNNs aim to learn discriminative latent representations from the data for downstream tasks. Among various neural architectures, the majority of works have mainly explored the usage of real-valued representations. Recently, an emerging research interest in complex [46], [47] and hypercomplex (i.e., quaternion) representations [30]–[32] has arisen. For example, Parcollet et al. successfully make the first attempt to devise a quaternion multi-layer perceptron for language understanding [48]. For image classification, two highly effective quaternion convolutional neural networks are proposed in [30] and [49], along with fundamental tools like quaternion-valued convolution operation, batch normalization,
and initialization. In the pursuit of maximum performance, both recurrent neural and convolutional networks are endowed with quaternion representations and achieve dominating results in different tasks, such as speech recognition [32], sound detection [50], and image conversion [31].

Apart from quaternion neural networks, quaternion representations also witness state-of-the-art performance in other areas like knowledge graph embedding [51] and collaborative filtering [33]. Notably, progress on quaternion representations for deep learning is still in its infancy [29], making its immense potential largely unexplored. In short, the benefit from quaternion representations can be attributed to several factors. Firstly, quaternions are intuitively linked to associative composition [29], leading to higher representation capacity and generalizability than real-valued counterparts [33]. Also, the hypercomplex Hamilton product provides a greater extent of expressiveness with a four-fold increase in interactions between real and imaginary components. Secondly, the asymmetry of Hamilton product has also demonstrated promising advantages in learning entity relationships in knowledge graphs [51] and inferring user-item preferences in recommender systems [33]. Thirdly, in the case of quaternion representations, due to the 75% parameter reduction in the Hamilton product, quaternion neural networks also enjoy a significant reduction in parameter size [29], [31], [32], and can alleviate the risk of overfitting brought by too many degrees of freedom in learnable parameters [49].

C. Network Compression

Our work is related to, but highly distinct from the research on network compression, among which two major representatives are network pruning [52] and quantization [53]. Network pruning reduces the weight parameters needed in neural models by cutting off unnecessary connections between neurons, while network quantization maps all continuous weight parameters into a fixed set of discrete values to minimize the total bits needed to store them [54]. However, those pruning and quantization methods are specifically designed to compress a well-trained large neural network, and this is non-comparable to our motivation of designing and training a lightweight prediction model from scratch. Furthermore, such network compression paradigms have not yet been utilized in FM-based models for predictive analytics, leaving it a promising direction for future research.

VII. Experiments

In this section, we first outline the evaluation settings regarding the CTR prediction task. Then, we conduct experiments to evaluate QFM and QNFM in terms of both effectiveness and efficiency. In particular, we aim to answer the following research questions (RQs) via experiments:

**RQ1:** How effectively can QFM and QNFM perform sparse predictive analytics compared with state-of-the-art FM-based models?

**RQ2:** How is the parameter efficiency of our models?

**RQ3:** How do the hyperparameters affect the performance of QFM and QNFM in CTR prediction?

**RQ4:** How do QFM and QNFM benefit from each component of the proposed model structure?

**RQ5:** How is the training efficiency and scalability of both models when handling large-scale data?

A. Datasets

To validate the performance of our proposed models in terms of CTR prediction accuracy, we adopt four real-world datasets. All datasets used in our experiment are in large scale and publicly available. The primary statistics are shown in Table I, and their properties are introduced below.

- **Trivago:** This is a web search dataset consisting of users’ visiting logs (e.g., city of the user, device being used, etc.) on different hotel booking webpages.
- **Criteo:** This is an industry-level benchmark dataset for CTR prediction. As our largest dataset, it has more than 45 million users’ clicking instances on displayed advertisements, with a mixture of 26 categorical feature fields and 13 numerical feature fields.
- **Avazu:** This benchmark dataset is similar to Criteo in size. It records mobile users’ behaviors including whether or not a displayed advertisement is clicked.

B. Baseline Methods

We briefly introduce the baselines for comparison below.

- **FM:** This is the original FM model [16] with proven effectiveness in many prediction tasks.
- **DeepCrossing:** It stacks multiple residual network blocks upon the concatenation layer for feature embeddings in order to learn deep cross features [8].
- **AFM:** The attentional FM [19] introduces an attention network to distinguish the importance of different pairwise feature interactions.
- **DCN:** The deep cross network [45] takes the outer product of concatenated feature embeddings to explicitly model feature interaction.
- **xDeepFM:** It stands for the extreme deep FM [21] that has a compressed interaction network to model vector-wise feature interactions for CTR prediction.
- **AutoInt:** Having a multi-head self-attentive neural network as the core component [15], it automatically learns the high-order interactions of input features.

**TABLE I**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Instance</th>
<th>#Field</th>
<th>#Feature (Sparse)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trivago</td>
<td>2,464,024</td>
<td>30</td>
<td>1,331,752</td>
</tr>
<tr>
<td>Criteo</td>
<td>45,840,617</td>
<td>39</td>
<td>998,960</td>
</tr>
<tr>
<td>Avazu</td>
<td>40,428,967</td>
<td>23</td>
<td>1,544,488</td>
</tr>
</tbody>
</table>

2http://www.recsyschallenge.com/2019/  
3http://labs.criteo.com/2014/02/  
4https://www.kaggle.com/c/avazu-ctr-prediction
- **HFM**: The holographic FM [55] uses circular convolutions for feature interaction modelling. The idea behind HFM is similar to ours, i.e., maximizing the pairwise feature interaction modelling with no additional parameters upon the plain FM.

- **HFM+**: This is the DNN-enhanced version of HFM proposed by [55].

Notably, FM, AFM and HFM are for second-order feature interaction modelling, while all other baselines account for higher-order interactions. Dedicated to CTR prediction, DeepCrossing, xDeepFM, DCN and AutoInt are currently the state-of-the-art methods in this line of research.

C. Evaluation Protocols

For each dataset, we randomly split its instances with a ratio of 8:1:1 for training, validation, and test, respectively. We adopt two well-established evaluation metrics for CTR prediction [8], [21], namely Area under the ROC Curve (AUC) and Log Error (LE, equivalent to cross-entropy in our case). Furthermore, as the output in this task is a probability that approximates the binary label \( \{0, 1\} \), we further adopt Root Mean Squared Error (RMSE) [14], [19]. AUC measures the probability that a positive instance will be ranked higher than the negative one. It only takes into account the order of predicted instances and is insensitive to class imbalance problem. In contrast, LE and RMSE straightforwardly evaluate the distance between the predicted possibility and the true label for each instance. It is worth noting that, in the context of CTR prediction where service providers commonly have a large user base, an improvement on the aforementioned metrics at the 0.001-level is regarded significant [15], [22], [45], [56].

D. Hyperparameter Settings

To be consistent, we report the overall performance of our models with \( d = 64 \) for QFM and a unified parameter set \( \{d = 64, l = 1, \rho = 0.1\} \) for QNFM. Note that as described in Section V-B, a 64-dimensional quaternion representation in QFM/QNFM actually takes up 256 bits to represent all the information. Details on the effect of different hyperparameter settings will be given in Section VII-G. For all baselines, we set the dimensions of both the embedding layer and hidden layer (if there is any) to 256 to ensure the same representation capacity as QFM/QNFM, and tune all remaining hyperparameters via grid search. Specifically, we adopt a 2-tier residual network in DeepCrossing, and the numbers of deep feature interaction layers in DCN, xDeepFM and AutoInt are all set to 3 while xDeepFM is also coupled with a 2-layer feed-forward network by default. For HFM+, we use its original setting of 3 feedforward layers. For both NFM and AFM, we implement a single-layer structure in their feed-forward network and attention network, respectively.

E. Effectiveness Analysis (RQ1)

We showcase all models’ performance on CTR prediction with Table II. Based on the experimental results, we draw the following observations.

First of all, our proposed QNFM, which is the neural network-enhanced version, consistently and significantly outperforms all the baseline methods. This verifies the efficacy of Hamilton product in modelling the intricate feature interactions. In particular, on the highly sparse Trivago dataset which has a similar total number of features as Avazu but 16 times less observed training instances, the quaternion-valued feature interaction scheme in QNFM can fully learn the contexts of interactions between available features, thus yielding 1.6% performance gain on AUC over the second best model (i.e., xDeepFM). Meanwhile, as we will demonstrate in Section VII-F, QNFM achieves the best prediction results with substantially lower parameter consumption compared with all state-of-the-art baselines.

Secondly, as a relatively shallow and simple model, QFM shows strong competitiveness in terms of prediction performance. For example, on Criteo and Avazu, QFM outperforms the majority of all baselines except for xDeepFM and AutoInt. It also produces a significant improvement over the plain FM on all three datasets with an average performance increase of 2.13%, 5.22%, 3.30% on AUC, LE, and RMSE, respectively.

Lastly, the results imply that deep neural network-based FM variants are generally more powerful in sparse predictive analytics. Though AFM utilizes a deep layer to compute the attention weight for every pairwise feature interaction, the combinatorial effect of all feature interactions are still merged in a linear fashion, leading to inferior performance among other deep FM-based models like DCN. Another interesting finding is that straightforwardly using deep networks for high-order feature interaction modelling does not necessarily lead to better prediction performance, which is evidenced by the
slightly lower AUC scores achieved by DeepCrossing and NFM on Avazu in comparison to FM and AFM.

**F. Parameter Efficiency Analysis (RQ2)**

As we target on high-performance FM-based models that can minimize the parameter costs, we compare all models’ parameter efficiency w.r.t. their performance. In Table III, we report the extra parameter size introduced by each model on top of the plain FM, as well as their performance gain over the FM versus the plain FM. The results are based on the same embedding and latent dimension of 256 in $\mathbb{R}$, which is equivalent to $d = 64$ in $\mathbb{H}$.

<table>
<thead>
<tr>
<th>Method</th>
<th>Extra Parameter Size vs. FM</th>
<th>AUC Performance Gain vs. FM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepCrossing [8]</td>
<td>$2.0 \times 10^2$ $2.6 \times 10^5$ $1.5 \times 10^6$</td>
<td>3.96% 1.44% -1.14%</td>
</tr>
<tr>
<td>NFM [14]</td>
<td>$6.6 \times 10^2$ $6.6 \times 10^5$ $6.6 \times 10^8$</td>
<td>3.68% 0.98% -0.08%</td>
</tr>
<tr>
<td>AFM [19]</td>
<td>$6.6 \times 10^2$ $6.6 \times 10^5$ $6.6 \times 10^8$</td>
<td>2.16% 0.48% 0.13%</td>
</tr>
<tr>
<td>DCN [45]</td>
<td>$2.1 \times 10^2$ $2.7 \times 10^5$ $1.7 \times 10^8$</td>
<td>4.73% 1.48% -0.38%</td>
</tr>
<tr>
<td>xDeepFM [21]</td>
<td>$2.1 \times 10^2$ $2.8 \times 10^5$ $1.6 \times 10^8$</td>
<td>6.32% 1.72% 0.44%</td>
</tr>
<tr>
<td>AutoInt [15]</td>
<td>$1.0 \times 10^5$ $1.0 \times 10^9$ $9.9 \times 10^{15}$</td>
<td>5.70% 2.05% 0.45%</td>
</tr>
<tr>
<td>HFM+ [55]</td>
<td>0 0 0</td>
<td>3.62% 1.53% -0.54%</td>
</tr>
<tr>
<td>QFM</td>
<td>0 0 0</td>
<td>5.84% 1.80% 0.61%</td>
</tr>
<tr>
<td>QNFM</td>
<td>$1.6 \times 10^1$ $1.6 \times 10^1$ $1.6 \times 10^5$</td>
<td>4.36% 1.67% 0.35%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.96% 2.14% 0.60%</td>
</tr>
</tbody>
</table>

**TABLE III**

*COMPARISON ON EXTRA PARAMETERS AND AUC PERFORMANCE GAIN VERSUS THE PLAIN FM. THE RESULTS ARE BASED ON THE SAME EMBEDDING AND LATENT DIMENSION OF 256 IN $\mathbb{R}$, WHICH IS EQUIVALENT TO $d = 64$ IN $\mathbb{H}$."

We answer RQ3 by investigating the performance fluctuations of QFM/QNFM with varied hyperparameters. Particularly, as mentioned in Section V-A, we study the impact of latent dimension $d$ on both QFM and QNFM. We further investigate QNFM’s sensitivity to the feed-forward network depth $l$ and the dropout ratio $\rho$. Based on the standard setting $\{d = 64, l = 1, \rho = 0.1\}$ of QNFM, we vary the value of one hyperparameter while keeping the others unchanged, and record the new prediction result achieved. We choose AUC and LE to show the performance differences while RMSE is omitted as it shows a similar trend as LE. Figure 2 lays out the results with different parameter settings.

**G. Hyperparameter Sensitivity (RQ3)**

As we target on high-performance FM-based models that can minimize the parameter costs, we compare all models’ parameter efficiency w.r.t. their performance. In Table III, we report the extra parameter size introduced by each model on top of the plain FM, as well as their performance gain over the FM versus the plain FM. The results are based on the same embedding and latent dimension of 256 in $\mathbb{R}$, which is equivalent to $d = 64$ in $\mathbb{H}$.

<table>
<thead>
<tr>
<th>Method</th>
<th>Extra Parameter Size vs. FM</th>
<th>AUC Performance Gain vs. FM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepCrossing [8]</td>
<td>$2.0 \times 10^2$ $2.6 \times 10^5$ $1.5 \times 10^6$</td>
<td>3.96% 1.44% -1.14%</td>
</tr>
<tr>
<td>NFM [14]</td>
<td>$6.6 \times 10^2$ $6.6 \times 10^5$ $6.6 \times 10^8$</td>
<td>3.68% 0.98% -0.08%</td>
</tr>
<tr>
<td>AFM [19]</td>
<td>$6.6 \times 10^2$ $6.6 \times 10^5$ $6.6 \times 10^8$</td>
<td>2.16% 0.48% 0.13%</td>
</tr>
<tr>
<td>DCN [45]</td>
<td>$2.1 \times 10^2$ $2.7 \times 10^5$ $1.7 \times 10^8$</td>
<td>4.73% 1.48% -0.38%</td>
</tr>
<tr>
<td>xDeepFM [21]</td>
<td>$2.1 \times 10^2$ $2.8 \times 10^5$ $1.6 \times 10^8$</td>
<td>6.32% 1.72% 0.44%</td>
</tr>
<tr>
<td>AutoInt [15]</td>
<td>$1.0 \times 10^5$ $1.0 \times 10^9$ $9.9 \times 10^{15}$</td>
<td>5.70% 2.05% 0.45%</td>
</tr>
<tr>
<td>HFM+ [55]</td>
<td>0 0 0</td>
<td>3.62% 1.53% -0.54%</td>
</tr>
<tr>
<td>QFM</td>
<td>0 0 0</td>
<td>5.84% 1.80% 0.61%</td>
</tr>
<tr>
<td>QNFM</td>
<td>$1.6 \times 10^1$ $1.6 \times 10^1$ $1.6 \times 10^5$</td>
<td>4.36% 1.67% 0.35%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.96% 2.14% 0.60%</td>
</tr>
</tbody>
</table>

We answer RQ3 by investigating the performance fluctuations of QFM/QNFM with varied hyperparameters. Particularly, as mentioned in Section V-A, we study the impact of latent dimension $d$ on both QFM and QNFM. We further investigate QNFM’s sensitivity to the feed-forward network depth $l$ and the dropout ratio $\rho$. Based on the standard setting $\{d = 64, l = 1, \rho = 0.1\}$ of QNFM, we vary the value of one hyperparameter while keeping the others unchanged, and record the new prediction result achieved. We choose AUC and LE to show the performance differences while RMSE is omitted as it shows a similar trend as LE. Figure 2 lays out the results with different parameter settings.
TABLE IV
ABLATION TEST RESULTS WITH DIFFERENT MODEL ARCHITECTURES, WHERE “↓” MARKS A SEVERE (OVER 2%) PERFORMANCE DROP.

<table>
<thead>
<tr>
<th>Model</th>
<th>Metric</th>
<th>Architecture</th>
<th>Trivago</th>
<th>Criteo</th>
<th>Avazu</th>
</tr>
</thead>
<tbody>
<tr>
<td>QFM</td>
<td>AUC</td>
<td>Default</td>
<td>0.8543</td>
<td>0.8025</td>
<td>0.7739</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dot Product-based FI</td>
<td>0.8217↓</td>
<td>0.7902</td>
<td>0.7708</td>
</tr>
<tr>
<td></td>
<td></td>
<td>One-Way FI</td>
<td>0.8476</td>
<td>0.7981</td>
<td>0.7721</td>
</tr>
<tr>
<td></td>
<td>LE</td>
<td>Default</td>
<td>0.4696</td>
<td>0.4492</td>
<td>0.3847</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dot Product-based FI</td>
<td>0.5154↓</td>
<td>0.4522</td>
<td>0.3860</td>
</tr>
<tr>
<td></td>
<td></td>
<td>One-Way FI</td>
<td>0.4724</td>
<td>0.4503</td>
<td>0.3852</td>
</tr>
<tr>
<td>QNFM</td>
<td>AUC</td>
<td>Default</td>
<td>0.8853</td>
<td>0.8062</td>
<td>0.7758</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pure Element-wise IP</td>
<td>0.8616↓</td>
<td>0.7879↓</td>
<td>0.7667</td>
</tr>
<tr>
<td></td>
<td></td>
<td>One-Way IP</td>
<td>0.8821</td>
<td>0.8024</td>
<td>0.7743</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Removing RC</td>
<td>0.8769</td>
<td>0.8010</td>
<td>0.7714</td>
</tr>
<tr>
<td></td>
<td>LE</td>
<td>Default</td>
<td>0.4419↓</td>
<td>0.4579↓</td>
<td>0.3905↓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pure Element-wise IP</td>
<td>0.4332</td>
<td>0.4489</td>
<td>0.3845</td>
</tr>
<tr>
<td></td>
<td></td>
<td>One-Way IP</td>
<td>0.4356</td>
<td>0.4523</td>
<td>0.3862</td>
</tr>
</tbody>
</table>

Impact of $d$. The latent dimension $d$ is studied in $\{4, 8, 16, 32, 64\}$. As shown in Figure 2.(a)-(b), $d$ has an apparent effect on the performance of QFM and QNFM. When $d$ is relatively small (i.e., $d = 4$ or $d = 8$), it limits the expressiveness of QFM and QNFM, leading to unsatisfactory prediction results on all three datasets. Both models generally benefit from a relatively large value of $d$, but the performance improvement tends to become less significant when $d$ reaches a certain scale, which is 16 in most of our cases.

Impact of $l$. We investigate the impact of $l \in \{1, 2, 3, 4, 5\}$ in QNFM. Though stacking more deep layers may intuitively help achieve better prediction performance, QNFM barely shows visible improvements in AUC and LE when $l$ increases from 1 to 3 with subtle (i.e., 0.0001-level) performance fluctuations. When $l$ reaches 4 or 5, the performance of QNFM clearly deteriorates due to overfitting issues caused by excessive deep layers. Hence, setting $l = 1$ in QNFM is the most economical option for obtaining advantageous performance with minimal training difficulties.

Impact of $\rho$. In this study, we vary the dropout ratio $\rho$ in $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. Figure 2.(d) suggests that the best results are obtained when $\rho$ is between 0.1 and 0.3. After increasing the value to 0.4 to 0.5, QNFM experiences a significant decrease in performance on Criteo and Avazu. This is because that the huge amount and diversity of instances in these two industry-level datasets can be viewed as an inherent regularization term during training, so a larger $\rho$ will result in underfitting.

H. Importance of Quaternion Components (RQ4)

To better understand the benefits from the major quaternion components proposed in our models, we conduct ablation test on different degraded versions of QFM and QNFM. Each variant removes or modifies one key component from the model, and the corresponding results on three datasets are reported in Table IV. Similar to Section VII-G, AUC and LE are used for demonstration. In what follows, we introduce these variants and analyze their effect respectively.

Dot Product-based Feature Interaction (FI). We replace the inner Hamilton product in QFM with the straightforward dot product between two quaternion cores in the same channel, i.e., Eq.(11) $\rightarrow r^i_i r_j + a^i_i a_j j + b^i_i b_j j + c^i_i c_j K$. On this occasion, QFM actually models pairwise feature interactions in a similar way to the plain FM, where the only difference is that QFM separates the interaction contexts into four channels. Without the infused expressiveness of Hamilton product, significant performance decrease is observed on three datasets, especially on Trivago which has the highest data scarcity.

One-Way Feature Interaction (FI). This is a variant of QFM that simplifies the two-way feature interaction paradigm in Eq.(10) to one-way feature interaction, i.e., Eq.(10) $\rightarrow \sum_{i=1}^{n} \sum_{j=i+1}^{n} (v^i_i \odot v^j_j)$ where feature $i$ and $j$ only interact with each other once. Consequently, QFM suffers from the obviously inferior performance, which shows that by taking advantage of the non-commutative property of Hamilton product, the two-way FI effectively enriches the learned interaction information for accurate prediction.

Pure Element-wise Interaction Pooling (IP). We use the pure element-wise vector multiplication as in [14] for QNFM instead of the element-wise Hamilton product to compute $\tilde{v}^o$, i.e., Eq.(14) $\rightarrow r \odot r_j + a \odot a_j j + b \odot b_j j + c \odot c_j K$. Compared with Hamilton product-based pooling that encodes more semantics of feature interactions, the simplistic element-wise IP yields worse performance, and a severe performance drop on LE (over 2%) is observed on three datasets.

One-Way Interaction Pooling (IP). In this QNFM variant, by setting Eq.(13) $\rightarrow \sum_{i=1}^{n} \sum_{j=i+1}^{n} (v^i_i \odot v^j_j)$, we downgrade the original asymmetrical quaternion IP operation to one-way IP. Though it can better model feature interactions than pure element-wise IP, there is still a decrease in prediction accuracy. Hence, our proposed asymmetrical quaternion IP is able to make full use of all the contexts from feature interactions, thus ensuring optimal performance in prediction tasks.

Removing Residual Connections (RC). We construct another variant of QNFM by removing the residual connections in the quaternion feed-forward network defined in Eq.(15). Apparenty, without residual connections, we find that the performance of QNFM gets worse on three datasets. Presumably this is because information in lower layers (i.e., the output generated by the pooling layer) cannot be easily propagated to the final layer, and such information is highly useful for making predictions, especially on our experimental datasets with a large amount of sparse features.

I. Training Efficiency and Scalability (RQ5)

Due to the importance of practicality in real-life applications of predictive models, we validate the training efficiency and scalability of both QFM and QNFM in this section. When all hyperparameters in the network are fixed, the training time for SeqFM is expected to be linearly associated with the
number of training samples. We vary the proportions of the training data in \( \{0.2, 0.4, 0.6, 0.8, 1.0\} \), and then report the corresponding time cost for the model training. It is worth noting that the Criteo dataset is used for scalability test since it contains the most instances. The training time w.r.t. the data size is shown in Figure 3. When the ratio of training data gradually extends from 0.2 to 1.0, the training time for QFM increases from \( 0.352 \times 10^4 \) seconds to \( 2.137 \times 10^4 \) seconds. Due to the slightly higher model complexity of QNFM, its training time grows from \( \{5.2\times10^3\} \) seconds. Due to the slightly higher model complexity of QNFM, its training time grows from \( 0.498 \times 10^4 \) seconds to \( 3.138 \times 10^4 \) seconds. The overall trend shows that both models’ dependency of training time on the data scale is approximately linear. Hence, we conclude that QFM and QNFM is scalable to even larger datasets.

VIII. CONCLUSION

In this paper, to mitigate the memory-inefficient nature of existing deep FM-based models, we make an innovative exploration on developing quaternion-valued FM variants by stepping into the hypercomplex space. We propose QFM and QNFM, two novel models that support the modelling of intricate feature interactions with heavily reduced parameter consumption. Experiments on industry-level CTR prediction datasets fully demonstrate that our quaternion FM-based models represent an elegant blend of state-of-the-art prediction performance and lightweight parameterization compared with their real-valued counterparts.

REFERENCES