Energy-Efficient Gaussian Processes Using Low-Precision Arithmetic

Nicolas Alder [1] **Ralf Herbrich** [1]

Abstract

The widespread use of artificial intelligence requires finding energy-efficient paradigms for the field. We propose to reduce the energy consumption of Gaussian process regression using low-precision floating-point representations. We explore how low-precision representations impact the results of Gaussian process regression and how data set properties, implementation approach, model performance, and energy consumption interact. Our findings show that a well-conditioned kernel matrix allows reducing the energy consumption by up to 89.01% for 98.08% of arithmetic operations with little to no impact on model performance. Our findings are relevant whenever one needs to invert a symmetric full-rank matrix.

1. Introduction

To keep up with the massive increase in AI-related workload, it is essential to make improvements related to power consumption (Debus et al., 2023). In this study, we suggest using low-precision Gaussian process regression (GPR) as a means of decreasing the power consumption of this AI method. GPR is typically used for small data sets where the prediction of uncertainty is of key importance. Enhancing the efficiency of these routine tasks has the potential to generate significant overall power savings. We investigate the connection between Gaussian process regression, arbitrary low-precision utilization, and power consumption.

Gaussian process regression is a mature model and a powerful tool for regression. The model (Rasmussen & Williams, 2006) has been widely adopted, as evidenced by its reception in the academic community and its inclusion in many libraries for industry use.

We propose low-precision Gaussian processes to reduce the power consumption of Gaussian process regression. While previous research (Maddox et al., 2022) has investigated the use of half-precision representations to fit large data sets, our focus is on implementing arbitrary precisions for typical library users that collectively contribute to power consumption. Our evaluation therefore concentrates on small-sized datasets that make up the large amount of daily workloads that these libraries perform. Adapting libraries to our approach can lead to significant efficiency gains and power reduction. Our low-precision approach can be directly utilized by many processors and GPUs that have inherent abilities to use smaller floating-point representations through SIMD or other native hardware implementations without the need for special hardware. For CPU-based implementations, high-level libraries like scikit-learn use LAPACK and BLAS for efficient, hardware-specific operations written in Fortran. These implementations, optimized for processors, support basic linear algebra and higher-level operations (e.g., Cholesky decomposition) and can be adapted or extended with a similar effort as current traditional methods. GPU-based libraries benefit from various native low-precision formats, making CUDA implementations straightforward. FPGAs can accommodate any arbitrary format in implementations. Finally, this work encourages chip designers to recognize that numerous applications benefit from adaptable numerical formats. However, low-precision floating-point representations can accumulate large round-off errors. Determining the appropriate low-precision representation is a non-trivial task, as the algorithm implementation, the chosen kernel, the specific data set, and the desired model performance interact with a specific numerical representation. It is not known what precision to use for reasonable model performance. Existing work from numerical linear algebra provides theory-guided upper error bounds of specific arithmetic operations and even subtasks in Gaussian process regression. However, the complexity of what precision delivers a reasonable model performance in an end-to-end perspective for Gaussian process regression with real data is only feasible through an empirical evaluation.

For Gaussian process regression using Cholesky decomposition, we achieved a power reduction of up to 88.94% for 83.27% to 87.43% of all operations compared to using double-precision floating-point representations. When using conjugate gradients, we achieved a decrease of up to 89.01% in energy consumption for 96.96% to 98.08% of

[1]Hasso Plattner Institute, Potsdam, Germany. Correspondence to: Nicolas Alder <nicolas.alder@hpi.de>, Ralf Herbrich <ralf.herbrich@hpi.de>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

all operations. Using low-precision representations leads to less than ± 0.02 deviation in root mean squared error of the test set and less than ± 0.04 deviation on the train set. The uncertainty calibration changes by less than ± 0.03.

Our contributions can be summarized as follows:

1. We evaluate the use of *arbitrary low-precision floating-point number representations for reducing power consumption* in Gaussian process regression.

2. We demonstrate that the energy usage of Gaussian process regression can be significantly decreased by utilizing low-precision representations, without compromising the model's performance. Our findings indicate a necessary *minimum precision* for model performance that depends on the condition number of the kernel matrix. Numerical instabilities at higher precisions can be alleviated by using *larger exponents* instead of increasing precision. This is also relevant for stabilizing computations when scaling Gaussian processes to larger datasets with high condition numbers. However, this paper does not focus on scaling, but on small-sized data.

3. We measure power at the circuit level instead of performing system-level benchmarks, providing *platform-independent relative power savings*.

4. We provide an energy-efficiency method for *inverting symmetric full-rank matrices* using conjugate gradients or Cholesky decomposition.

All source code is available at `https://github.com/nicolas-alder/energy-efficient-gps`

2. Related Work

2.1. Scalable Gaussian Process Regression

Gaussian process regression is often implemented with the Cholesky decomposition to calculate the inverse of its kernel matrix. However, the cubic complexity of the Cholesky decomposition prohibits the use of Gaussian process regression for large data sets. The field of scalable Gaussian process regression efficiently combines hardware, approximation, and stochastic concepts to push computational limits for large datasets (Titsias, 2009; Wilson & Nickisch, 2015; Pleiss et al., 2018; Gardner et al., 2018; Wang et al., 2019; Maddox et al., 2022). This field is not focused on energy consumption, but on computational effort savings and distribution for large datasets. However, these two goals can be related. A recent study by Maddox et al. (2022) found that a modified half-precision conjugate gradients algorithm often produces comparable predictive means. In addition to the Cholesky decomposition approach, we also included Maddox et al.'s modified conjugate gradients algorithm (2022) in our evaluation.

2.2. Numerical Linear Algebra

Upper round-off error bounds are known for floating-point representations and corresponding arithmetic operations. Numerous works (Martin et al., 1965; Wilkinson, 1966; Meinguet, 1983; Kiełbasiński, 1987; Sun, 1992; Higham, 1990) have been published since the 1950s to examine bounds for the Cholesky decomposition. Recent work also investigated the conjugate gradients method (Greenbaum et al., 2021). Although this research provides important information about the error ingredients for empirical evaluation, the worst-case bounds are not specific enough to guide the construction of real-world Gaussian process models.

2.3. Efficient Hardware Acceleration

Certain frameworks are designed to focus specifically on Gaussian processes and allow efficient GPU utilization (Gardner et al., 2018; Matthews et al., 2017). Furthermore, due to low energy consumption and the possibility of highly efficient special-purpose design, a whole community has been focused on efficient linear algebra operations and algorithms in FPGAs (Gonzalez & Núñez, 2009; Yang et al., 2010; Roldao & Constantinides, 2010; Korcyl & Korcyl, 2019; Naher et al., 2019; Malakonakis et al., 2022; Song et al., 2022; 2023). We also use the design-flexibility and instrumentation options of an FPGA for the power consumption measurements. In contrast to the aforementioned works, our goal is to measure the relative power savings achieved by implementing arithmetic operations at the circuit level, without being specific to any particular hardware platform. Compared to specialized hardware accelerators, we assume that improving algorithms for arithmetic operations (and corresponding circuits) rather than specific platforms is more enduring. Thus, savings apply to any generic platform that uses the respective circuits.

3. Approach

This section presents the necessary ingredients for our approach: the Gaussian process model, low-precision representations, and assumptions for the power benchmark.

3.1. Gaussian Process Regression

We investigate Gaussian process regression $f(\mathbf{x}) \sim \mathrm{GP}(m(\mathbf{x}), \mathrm{cov}(\mathbf{x}, \mathbf{x}'))$ with noise-free data $y = f(\mathbf{x})$ for an input vector $\mathbf{x}$ with mean $m(\mathbf{x}) = 0$ according to the definition of Rasmussen and Williams (Rasmussen & Williams, 2006). The covariance $\mathrm{cov}(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}')$ is often given

by the Radial Basis Function (RBF)

$$\mathrm{k}_{\mathrm{RBF}}(\mathbf{x}, \mathbf{x}') = \sigma^2_{\text{output scale}} \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2l^2}\right). \quad (1)$$

For a matrix of training features X, training target vector $\mathbf{y}$ and test feature matrix X_*, we obtain the predictive mean vector $\bar{\mathbf{f}}_*$ by

$$\bar{\mathbf{f}}_* \triangleq \mathbb{E}[\mathbf{f}_* | X, \mathbf{y}, X_*] = K_{X_*,X}[K_{X,X}]^{-1}\mathbf{y} \quad (2)$$

and predictive covariance matrix $\mathrm{cov}(\mathbf{f}_*)$ by

$$\mathrm{cov}(\mathbf{f}_*) = K_{X_*,X_*} - K_{X_*,X}[K_{X,X}]^{-1}K_{X,X_*}\,, \quad (3)$$

where $K_{X,\tilde{X}}$ is a matrix of covariances $k(\mathbf{x}, \tilde{\mathbf{x}})$ for all $\mathbf{x}$ in X and $\tilde{\mathbf{x}}$ in $\tilde{X}$, respectively.

In addition to matrix multiplication and subtraction, inversion of the quadratic kernel matrix $[K_{X,X}]^{-1}$ is the most expensive operation of Gaussian process regression. Reformulated as a system of linear equations, it is commonly implemented by solving the Cholesky decomposition in $O(n^3)$ for $K_{X,X} \in \mathbb{R}^{n \times n}$. Then forward and backward substitution is applied in $O(n^2)$ for each desired matrix-vector product. The Cholesky decomposition is a type of LU factorization, where $U = L^T$. It exploits the symmetry and positive definiteness of the kernel matrix and decomposes the kernel matrix K into a lower triangular matrix L and an upper triangular matrix L^T, such that $L \cdot L^T = K$.

We use the Cholesky-Banachiewicz algorithm (Banachiewicz, 1938) to build the lower triangular matrix row by row. Each lower triangular element is derived by first taking the equivalent element from the original matrix. If it is a diagonal element, we subtract the sum of the squared elements in the same row of L left to the diagonal element before taking the square root of the term. If it is not a diagonal, we subtract the sum of the products of each element of the same row to the left of L and the element of the same column from the above row. We divide the term by the diagonal element of the same column. Intuitively, this process is sometimes compared to completing the square for polynomials or calculating the square root of a matrix. The vast amount of arithmetic operations consists of multiplications and additions and depends on the size of the kernel matrix. We apply the Cholesky decomposition according to Rasmussen & Williams (2006).

Alternatively, the conjugate gradients method can be employed to numerically solve for the matrix-vector product of the linear system of equations. The inverse matrix product is estimated iteratively, resulting in computational savings that depend on the number of iterations required to achieve an acceptable approximation. This approach was previously employed by Maddox et al. (2022) for half-precision Gaussian processes to enable the processing of very large datasets. Furthermore, they modified the conjugate gradient algorithm to be more robust against round-off errors. First, they reduced the chance of exponent overflows by rescaling the kernel matrix-vector multiplications. Second, they applied the logsumexp trick to the step size and conjugacy terms to improve round-off error in calculations (smaller absolute values computed with higher precisions in float-point formats) and to reduce overflows. Third, they reorthogonalized the residual vector with respect to previous residual vectors in each iteration to address round-off impact. Together with the Choleyky decomposition, we included their modified version of the conjugate gradients algorithm in our evaluation. The Appendix provides a detailed discussion of both algorithms, including pseudocode and error bounds derived from numerical linear algebra.

3.2. Low-Precision Floating-Point Representations

Gaussian process regression is commonly implemented with 64-bit double-precision floats according to the IEEE-754 standard. Consequently, a sign bit $\pm$, a significand with t precision bits $d_1, d_2, \ldots, d_t$, and an exponent e with bias $b = 2^{e-1} - 1$ constitute a normalized floating-point representation of x via

$$x = \pm 2^{e-b} \cdot d_1.d_2...d_t, \quad (4)$$

where

$$d_i \in \{0, 1\} \text{ and } d_1 = 1. \quad (5)$$

The IEEE double-precision format uses a total of $t = 53$ bits for precision, one bit being implicit due to normalization, and 11 bits for the exponent. The precision component determines the smallest unit between two real numbers that can be represented in the calculations. When the actual value deviates from its representation, a round-off error occurs. In terms of relative error, double-precision has a maximum round-off of 2^{-52}. The exponent component determines the range of a represented number. However, as real numbers of larger magnitude still have the same number of precision bits to represent this range, the potential for absolute error increases due to the uneven spacing between neighboring representations. With every arithmetic operation, a new round-off error might be introduced and accumulates. The IEEE standard specifies round-ties-to-even that cancel the round-off error for a Gaussian error distribution in expectation. The presence of normally distributed errors (of equal magnitudes) is therefore critical for employing lower precisions as they directly correspond to round-off accumulation and performance degradation.

Accumulated round-off errors caused by low-precision computations can have a significant impact on the performance of Gaussian process regression. This can result in a notice-

able difference between the predictive mean and the covariance compared to calculations done with double-precision. Additionally, large round-off errors can make intermediate results unstable and cause them to fail, potentially leading to overflow.

Although the works (Martin et al., 1965; Greenbaum et al., 2021) from numerical linear algebra do not guide the selection of a specific precision for the Cholesky decomposition or conjugate gradient, they offer information on what has an impact on the accumulated error. The eventual error in our Gaussian process model is influenced by various factors, including the precision, the size of the kernel matrix (i.e., the size of the training data) and the conditioning of the kernel matrix. It is important to note that a larger kernel matrix does not necessarily result in worse conditioning. However, in our preliminary experiments, we observed that condition numbers tend to increase for larger sampling sizes of the same data set. Additionally, the conditioning of a kernel matrix is highly dependent on the specific characteristics of the data set and the kernel function used, as well as its hyperparameters. To understand these complex relationships, we employed an empirical evaluation approach, where we assessed the effects of the low-precision methodology.

3.3. Energy Consumption

To determine power consumption and potential savings in a generic and platform-independent manner, we focused on comparing relative reductions rather than absolute values. During our experiments, we counted the arithmetic operations that were performed while fitting the training data and during the inference phase. We then established a relationship between these operations and the power consumption of the corresponding arithmetic circuits, taking into account the precision used. To determine the extent of power savings, we performed a benchmark of the energy usage of simplified arithmetic circuits at different precision levels at the circuit level. Simplified refers to the implementation of integer-based circuits as they exhibit the same scaling properties as their complex IEEE floating-point counterparts. Given that the majority of operations in Gaussian process regression involve addition and multiplication or can be traced back to it, we focused on implementing these two operations on circuits and measuring their power consumption.

We implement a Carry-Ripple Adder circuit (by using a 2s complement, subtraction is equivalent to using an addition circuit). The state machines used in (non-)restoring division and add-and-shift multiplication have very similar designs and exhibit the same scaling properties. For simplicity, we assume that the power consumption of all other operations is independent of a numeric representation. Thus, we can assign any numerical operation in GPR to addition, multiplication, division, or being constant (not optimized). The energy reduction is the weighted mean of the power measurements for a representation, depending on the fraction of additions/subtractions and multiplications/divisions.

As we only benchmark arithmetic operations, our measurements do not include power consumption for memory transfer and bus traffic. The circuit size complexity of an arithmetic circuit is a significant factor in determining its relative power consumption, since it quantifies the resources based on the size of the numeric representation. Therefore, the choice of circuit design can impact potential power savings. Although alternative circuit designs might be employed in practical hardware, our method remains advantageous as long as the overall resource demands (circuit-size or required cycles) are minimized by using smaller formats. Our addition and multiplication circuits have linear complexity, which represents very moderate scaling. Although hardware manufacturers do not disclose their circuit designs, many modern designs exhibit quadratic or higher complexities. Therefore, our measurements only represent a conservative estimate of power savings; it is likely that the actual power savings are higher than what we report here.

4. Experimental Setup

Our experimental setup includes software-based experiments for estimating performance and tracking arithmetic operations, as well as hardware-based experiments for benchmarking power consumption of arithmetic operations.

4.1. Software Benchmarks

We utilized the GMPY2[1] library to implement Gaussian process regression in Python, which supports arbitrary-precision floating-point representations. All operations within Gaussian process regression were wrapped to monitor the number of calls made.

We selected six different regression datasets for experimentation. Five of them were arbitrarily chosen from the Penn Machine Learning Benchmark (Romano et al., 2021) to include diverse dataset properties. The sixth dataset, the California Housing dataset (Pedregosa et al., 2011), was specifically chosen to introduce a dataset that produces a very ill-conditioned kernel matrix. Although our analysis is restricted to the selected datasets and their characteristics, exploratory testing on additional datasets aligned with our findings and suggests that the condition number is a crucial metric for understanding the relationship between model performance and the minimum precision that can be used for the algorithms. In typical library workflows, the RBF length-scale hyperparameter is often automatically determined and is crucial for the performance and learning capabilities of the resulting model. To ensure the validity of

[1] www.mpfr.org

a real-world setting in our experiments, we used the BFGS algorithm, which is also employed in scikit-learn, to determine the length-scale l in (1). The length-scale optimization was performed with double precision and was not included in our operations tracking. We assumed that the datasets have an output variance of 1.

From each dataset, we have randomly drawn 500 samples as a train set and 500 samples as a test set. For each dataset, we evaluated mantissa precisions with $t = 3, 4, \ldots, 8$ in one-bit increments and $14, 24, 34, 44, 53$ in 10 bit increments up to double-precision. The exponent e was always kept at 11-bits. We repeated every experiment fives times and reported means and standard deviations. The differences reported refer to means. Equally, the conditioning of the kernel matrix is reported as most computations occur in the kernel matrix inversion. If not all five experiment repetitions are successful due to instability, we additionally report the number of successful experiment repetitions.

We performed 20 iterations to compute the predictive mean using the conjugate gradient approach. For the predictive covariance of each test point, we used 5 iterations. In the case of the ill-conditioned California Housing dataset, we used 100 iterations for the predictive mean. The number of iterations was determined by conducting exploratory experiments in double-precision, and we chose the minimum number of iterations that produced reasonable performance.

To evaluate the performance of the predictive mean model, we use the root mean squared error (RMSE)

$$\mathrm{RMSE}(\mathbf{y}, \hat{\mathbf{y}}) = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}, \tag{6}$$

with n being the number of test points and $\hat{y}_i$ representing the model's prediction compared to the ground truth y_i.

The predictive (co)variance provides an estimate $\hat{v}$ of how uncertain the model predictions are. To assess the quality of the uncertainty estimates, we propose a metric that evaluates different confidence interval levels (ranging from 1 to 99) against the fraction of actual hits in the test data. The deviation from the perfect calibration is then averaged to arrive at the uncertainty calibration (UC) for the test point predictions $\hat{\mathbf{y}}, \hat{\mathbf{v}}$, when the actual values are $\mathbf{y}$:

$$\mathrm{UC}(\mathbf{y}, \hat{\mathbf{y}}, \hat{\mathbf{v}}) = \sum_{c=1}^{99} \left| \frac{c}{100} - \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\left(-z_c \leq \frac{y_i - \hat{y}_i}{\sqrt{\hat{v}_i}} \leq z_c \right) \right|. \tag{7}$$

The confidence boundaries are obtained by multiplying the standard deviation of a test prediction's covariance v with the z-score z_c for a respective confidence level c.

4.2. Hardware Benchmarks

The power consumption was measured by implementing an add-and-shift multiplier circuit and Carry-Ripple adder circuit in VHDL on an FPGA core. The FPGA board contained a dedicated power rail solely for the core and an onboard power measurement chip. We used a Digilent Genesys2 with Xilinx Kintex-7 (XC7K325T-2FFG900C) FPGA-Core and Texas Instruments INA219 chip as power monitor.

We carried out experiments to measure the power consumption depending on the precision of our arithmetic circuits. In order to ensure that our experiments are comparable and meaningful, we utilized a fixed-size pseudorandom (deterministic) uniform input distribution for the circuits. This input distribution emulates the inputs of the arithmetic circuits. It is important to note that the input distribution and the order of the sequential inputs can affect power consumption. If the input numbers are too close to each other, fewer capacitors may be loaded or released, leading to a different power consumption measure. To prevent such correlated power footprints, we ensured a uniform and constantly changing input distribution. We have developed circuits that operate using a clocked signal instead of being combinatorial. This ensures that an equal number of arithmetic operations are performed in each experiment, enabling meaningful comparisons between them. We implemented multiple arithmetic circuits in parallel to achieve a measurable power difference. We separately benchmarked our experimental setup, which included a bus system, a memory interface, and a microprocessor to read the power measurement registers. The power measurements have been adjusted to remove the power consumption of the experimental setup, reflecting only the arithmetic circuits' operations. Due to the labor-intensive nature of the experiments, we physically measured bit-size representations of 4, 14, 24, 34, 44, 53, 54, and 64 bits. In the software-based experiments, we always use a fixed level of precision. To link this precision with the power consumption measured in hardware experiments, we used a linear regression approach. This approach is reasonable due to the linear circuit-size complexity, as depicted in Figure 1. We always use a double-precision reference representation (53-bits) for comparison.

5. Results

This section begins by presenting the conditioning of the kernel matrices that we obtained for our datasets. We proceed to analyze the performance measures in relation to three observed phenomena: the necessary minimum precision, computational stability, and power consumption. The experimental results for this section are presented in Tables 1 for the conjugate gradient and 2 for the Cholesky decomposition approach. The complete results are in the Appendix.

Table 1. The impact of low-precision numeric representations on energy consumption and model performance in Gaussian Process Regression with conjugate gradient implementation. The table includes selected results: lowest precision with stable computations, lowest precision with competitive (bold) performances (Δ UC, Train and Test RMSE), and double-precision. California Housing (ch) experiments are listed in full. Brackets in Precision column indicate the number of stable experiments to total experiments.

	Data	Precision	Conditioning	Δ Train	Δ Test	Δ UC	% Operations	Δ Energy
Conjugate Gradient	fr	3	10 ± 3	0.50	0.03	-0.14		-98.38%
		8	$\mathbf{9 \pm 1}$	**0.01**	**0.00**	**0.00**	**97.99%**	**−89.01%**
		53	9 ± 1	-	-	-		-
	wn	3 (2/5)	80 ± 37	56.08	49.27	0.64		-98.38%
		8	$\mathbf{65 \pm 11}$	**0.04**	**0.01**	**−0.02**	**97.88%**	**−89.01%**
		53	79 ± 40	-	-	-		-
	st	4	96 ± 13	0.88	0.50	-0.28		-91.21%
		8	$\mathbf{149 \pm 36}$	**0.02**	**0.01**	**−0.01**	**97.27%**	**−88.98%**
		53	152 ± 54	-	-	-		-
	mv	3 (4/5)	131 ± 26	1.53	0.89	-0.23		-98.38%
		8	$\mathbf{146 \pm 63}$	**0.02**	**0.01**	**−0.01**	**97.99%**	**−89.01%**
		53	220 ± 190	-	-	-		-
	pl	4 (4/5)	$6,758 \pm 2,368$	4.17	3.34	-0.07		-91.20%
		8	$\mathbf{8,274 \pm 5,364}$	**0.04**	**0.00**	**−0.03**	**96.96%**	**−88.97%**
		53	$10,842 \pm 8,206$	-	-	-		-
	ch	4 (1/5)	$1,156,326 \pm 0$	144.37	144.82	0.59		-91.25%
		5 (4/5)	$3,483,705 \pm 1,927,678$	2.82	2.77	0.33		-94.64%
		14	$\mathbf{3,394,682 \pm 2,659,590}$	**−0.03**	**−0.02**	**0.01**	**98.08%**	**−75.99%**
		24 (4/5)	$3,647,790 \pm 2,357,889$	-0.03	0.00	-0.01		-70.19%
		34 (4/5)	$2,878,770 \pm 1,136,307$	-0.02	-0.01	0.00		-40.11%
		44	$2,526,487 \pm 1,514,300$	-0.03	0.01	-0.01		-22.22%
		53	$4,818,647 \pm 3,902,685$	-	-	-		-

5.1. Kernel Matrix Conditioning

The performances and stability of the experiments are largely influenced by the conditioning of the kernel matrix, since most arithmetic operations in the Gaussian process regression are part of the kernel matrix inversion. Listed in Tables 1 and 2 are results for the datasets Fried (fr), Wind (wn), and Satellite (st) which are well conditioned (a condition number less than or equal to $1e3$), Pol (pl) which is moderately conditioned (a condition number of less than or equal to $2e4$), and California Housing (ch) which is ill-conditioned (a condition number of less than or equal to $1e7$). Experiments with good conditioning consistently allowed us to use lower precisions without reducing the model performance or only observing minor effects, while ill-conditioned kernel matrices needed higher precisions for competitive performances or displayed numerical instabilities. The 'pl' dataset showed mixed behavior, depending on the implementation approach used.

We report the differences Δ in Tables 1 and 2 for RMSE on the train and test set and the uncertainty calibration on the test set only. The RMSE on the train set can be interpreted as an indicator of how close the computational result of a precision is to double-precision. The RMSE and uncertainty calibration deliver information if a numeric offset is also translated into test set error, which is most relevant in real-world settings.

5.2. Minimum precision

The experiments showed that both the Cholesky decomposition and conjugate gradient implementation approaches produce unstable computations and significantly worse model performance when precision is too low. When the precision met a minimum threshold (cf. bold precisions in Tables 1 and 2), the model's performance showed minimal or no difference compared to calculations at double-precision. The minimum precision threshold depends on the approach used for implementation and the conditioning of a dataset's kernel matrix. The data set, the chosen kernel (RBF), and the hyperparameters (optimized through BFGS) significantly influence the conditioning of the kernel matrix. Interestingly, our experiments showed that both implementation approaches required a minimum precision of 8 bits for well-conditioned matrices (cf. bold 8 bit precisions in Table 1 and 2). Interestingly, sometimes even RMSE even improved, possibly through implicit regularization. The RMSE differences on the train set were also low (≤ 0.04). The uncer-

Table 2. The impact of low-precision numeric representations on energy consumption and model performance in Gaussian Process Regression with Cholesky decomposition implementation. The table includes selected results: lowest precision with stable computations, lowest precision with competitive (bold) performances (Δ UC, Train and Test RMSE) and double precision. Brackets in Precision column indicate the number of stable experiments to total experiments.

	Data	Precision	Conditioning	Δ Train	Δ Test	Δ UC	% Operations	Δ Energy
Cholesky Decomposition	fr	3 (4/5)	10 ± 2	0.12	0.04	-0.09		-98.30%
		8	$\mathbf{9 \pm 1}$	**0.01**	**−0.02**	**0.01**	**87.02%**	**−88.94%**
		53	9 ± 2	-	-	-		-
	wn	3 (3/5)	57 ± 13	0.30	0.06	-0.15		-98.26%
		8	$\mathbf{54 \pm 11}$	**0.02**	**−0.02**	**0.01**	**86.52%**	**−88.90%**
		53	59 ± 9	-	-	-		-
	st	3 (4/5)	107 ± 7	0.27	0.07	-0.23		-98.06%
		8	$\mathbf{111 \pm 24}$	**0.02**	**−0.01**	**0.01**	**84.24%**	**−88.73%**
		53	114 ± 20	-	-	-		-
	mv	4 (4/5)	135 ± 39	0.13	0.00	-0.12	87.02%	-91.16%
		8	$\mathbf{192 \pm 55}$	**0.01**	**0.00**	**−0.01**	**87.02%**	**−88.94%**
		53	127 ± 72	-	-	-		-
	pl	**14**	$\mathbf{25,093 \pm 39,926}$	**0.00**	**0.00**	**−0.01**	**83.27%**	**−75.16%**
		24 (4/5)	$66,100 \pm 107,969$	0.00	0.00	0.00		-71.09%
		53 (4/5)	$12,815 \pm 10,215$	-	-	-		-
	ch	24	$4,574,294 \pm 2,490,886$	0.00	0.05	-0.01		-70.30%
		34	$\mathbf{2,856,070 \pm 1,929,682}$	**0.00**	**0.01**	**−0.03**	**87.43%**	**−40.02%**
		53	$3,222,575 \pm 1,872,118$	-	-	-		-

tainty calibration was maintained or improved (≤ 0.01).

We observe that even using 3-bit and 4-bit precision for datasets 'fr' and 'wn' partly resulted in a reasonable test set error (cf. Tables 1, 2 and Appendix). However, we consider this not a robust approach as the train differences are significantly higher (cf. "Δ Train" in Table 2 for 'fr' and 'wn' with 3 bits precision).

When working with the ill-conditioned 'ch' dataset, using an 8-bit representation with conjugate gradients leads to significantly more error compared to double-precision. For a comparable model performance, at least 14 bits are necessary. For Cholesky decomposition, 24 bits are sufficient for numerically stable results, while 34 bits ensure competitive model performance. Similar observations hold for the 'pl' dataset. When using conjugate gradients, 8 bits are sufficient for reasonable performance, but with Cholesky decomposition, 14 bits are the lowest usable precision, as 8 bits do not produce numerically stable computations.

In summary, the modified conjugate gradient approach surpasses the Cholesky decomposition in terms of exhibiting a lower minimum precision required. We found that using significantly lower precisions than double-precision resulted in little to no performance degradation for both implementation approaches and all kernel matrix conditionings. Sometimes, generalization even improved due to implicit regularization. Also, the effectiveness of using low-precision number representations hints at the presence of Gaussian-distributed errors that are compensated by round-ties-to-even.

5.3. Numerical Stability of Ill-Conditioned Datasets

Our experiments on the 'ch' dataset using the conjugate gradient approach demonstrate that higher condition numbers reduce the numerical stability of experiments, leading to an increase in overflows. We also find that precisions below the minimum threshold result in unstable experiments or poor model performance (as evidenced by 4 and 5-bit precisions for the 'ch' dataset in Table 1). Additionally, we observe more numerical instabilities at higher precisions (as seen with 24 and 34-bit precisions for 'ch' dataset in Table 1). Although demonstrating competitive model performance once stabilized, these experiments indicate that instability and performance have different underlying causes.

We increased the available exponent bits from 11 to 15 and conducted additional experiments on the 'ch' dataset with conjugate gradient and the 'pl' dataset with Cholesky decomposition for unstable experiment precisions (cf. results in the Appendix). As a result, most observed instabilities disappeared. This indicates that precision is not the only factor contributing to unstable computations and round-ties-to-even effectively mitigates round-off errors in computations beyond the minimum precision. The ill-conditioning of the kernel matrix leads to greater fluctuations in absolute values. Therefore, competitive results can be achieved by using larger exponents. When working with large or highly ill-

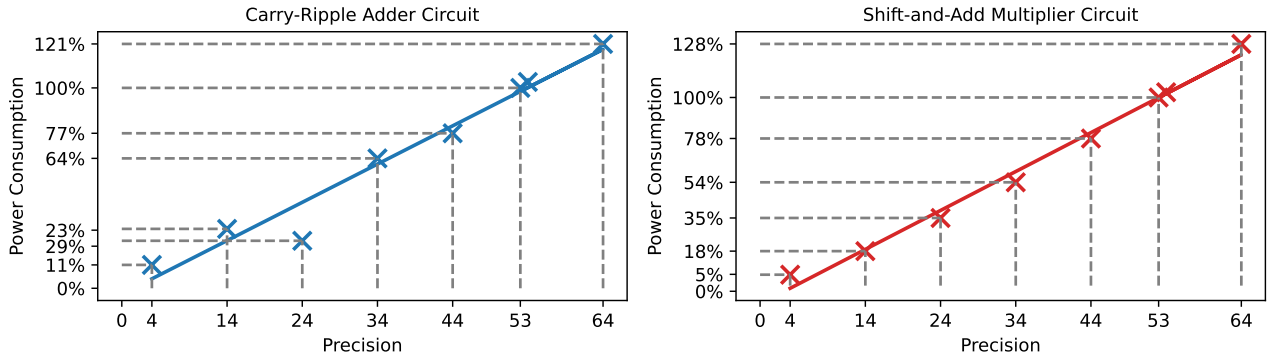

Figure 1. Results of the power consumption benchmark for addition and multiplication circuits relative to 53-bit double-precision, showing a linear power pattern in line with their circuit-size complexities. The power measurements were conducted on an FPGA-core.

conditioned datasets in (scalable) Gaussian process regression scenarios, this discovery holds particular significance. Large datasets and ill-conditioned kernel matrices can cause stability issues with double-precision calculations. However, increasing the exponent can help address this problem and result in more efficient processing compared to frequently utilized larger precisions.

In summary, ill-conditioned kernel matrices can lead to numerical instability, which can be efficiently addressed by using larger exponents. However, the impact of precision on stability is minor, as long as the minimum precision is met. Again, this suggests normally distributed errors for computations above the minimum precision. The use of larger exponents, instead of larger precisions, is particularly useful when efficiently scaling Gaussian processes for very large or ill-conditioned datasets.

5.4. Power Consumption

Lowering the precision of calculations during kernel matrix inversion using the conjugate gradient method can lead to a significant reduction in energy consumption. For well-conditioned datasets, this reduction can be as high as 89.01%, while for ill-conditioned datasets it can be up to 75.99%. This approach can reduce energy consumption for 96.96% to 98.08% of all operations involved in Gaussian process regression. Utilizing Cholesky decomposition can significantly reduce energy consumption by up to 88.94% for well-conditioned datasets and 40.02% for ill-conditioned datasets. However, the requirement of higher minimum precisions for the Cholesky decomposition approach results in reduced relative power savings. The fraction of optimized operations is slightly lower, comprising only 83.27% to 87.43% of all operations.

Figure 1 displays the benchmarking results for power consumption for addition and multiplication. Both circuits show a linear power consumption pattern, which is consistent with the expected behavior for circuits with linear size complexity. Anomalies in power consumption are observed in the 24-bit addition circuit, but these may be due to an efficient compiler mapping to our FPGA core, rather than a general rule. All other measurements for both circuits conform closely to the regression line.

In summary, reducing precision in Gaussian process regression leads to lower power consumption for most arithmetic operation calls. The relationship between precision and power consumption is set by the circuit-size complexity of arithmetic circuits. Although we opted for low-complexity circuits in our study, real-world arithmetic circuits usually have quadratic or cubic complexities to achieve faster throughput. Consequently, the associated power reductions in such circuits would be considerably higher.

6. Conclusion

It is possible to perform Gaussian process regression using floating-point operations with significantly lower precisions than the commonly used double-precision. This can be achieved without a significant decrease in predictive test performance (less than or equal to 0.02 RMSE) or uncertainty calibration (less than or equal to 0.03). Our experiments have shown that this approach can result in substantial power savings (up to 89.01%) for most operations (up to 98.08%). Also, power savings are proportional to the complexity of the utilized arithmetic circuit.

Based on our experiments, we have found that in order to achieve comparable model performance to that of double-precision, maintaining a minimum level of precision is necessary. Our findings indicate that well-conditioned datasets of small sizes require a minimum precision of 8 mantissa bits. Ill-conditioned datasets need 34 mantissa bits.

We have observed that the minimum precision required is heavily dependent on the condition number of the kernel matrix. Large kernel matrix condition numbers can cause instability problems, even when the minimum precision is met. However, there is an effective way to mitigate these instabilities by increasing the exponents instead of increasing the precision, which is also more energy efficient. This method is particularly helpful when dealing with large datasets or ill-conditioned kernel matrices in Gaussian process regression. Based on experimental results, there is evidence that computations beyond the minimum precision follow a Gaussian error distribution.

Our future research will investigate how the condition number is related to the error distribution. This will allow us to further reduce the required minimum precision and enhance the stability of ill-conditioned kernel matrix computations. Furthermore, an automated method is needed to determine the minimum required precision level based on the implementation and condition number, enabling easy adoption in libraries.

Acknowledgements

The authors acknowledge the financial support by the German Federal Ministry for Education and Research (BMBF) within the project KI-Servicezentrum Berlin Brandenburg 01IS22092.

Impact Statement

The aim of this research is to advance the field of Machine Learning and tackle the problem of energy consumption and energy-efficient approaches in AI. The potential societal implications include a decrease in CO_2 emissions and greater accessibility to cost-effective products.

References

Banachiewicz, T. Méthode de résolution numérique des équations linéaires, du calcul des déterminants et des inverses, et de réduction des formes quadratiques. *Bulletin International de l'Academie des Sciences de Pologne*, pp. 393–401, 1938.

Bunch, J. and Kaufman, L. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation*, 31(137):163–179, 1977.

Chronopoulos, A. and Gear, C. W. S-step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics*, 25(2):153–168, 1989.

Crout, P. A short method for evaluating determinants and solving systems of linear equations with real or complex coefficients. *Electrical Engineering*, 60(12):1235–1240, 1941.

Davies, A. *Efficient Implementation of Gaussian Process Regression for Machine Learning*. PhD thesis, Department of Engineering, University of Cambridge, 2005.

Debus, C., Piraud, M., Streit, A., Theis, F., and Götz, M. Reporting electricity consumption is essential for sustainable ai. *Nature Machine Intelligence*, 5(11):1–3, 2023.

Gardner, J., Pleiss, G., Weinberger, K., Bindel, D., and Wilson, A. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2018.

Ghysels, P. and Vanroose, W. Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. *Parallel Computing*, 40(7):224–238, 2014.

Gibbs, M. and Mackay, D. Efficient implementation of gaussian processes. Technical report, Cavendish Laboratory, Cambridge, UK, 1997.

Golub, G. and Loan, C. *Matrix Computations*. JHU press, 4th edition, 2013.

Gonzalez, J. and Núñez, R. LAPACKrc: Fast linear algebra kernels/solvers for FPGA accelerators. *Journal of Physics: Conference Series*, 180(1):012042, 2009.

Gratton, S., Simon, E., Titley-Peloquin, D., and Toint, P. Minimizing convex quadratics with variable precision conjugate gradients. *Numerical Linear Algebra with Applications*, 28(1), 2021.

Greenbaum, A. Behavior of slightly perturbed lanczos and conjugate-gradient recurrences. *Linear Algebra and its Applications*, 113:7–63, 1989.

Greenbaum, A., Liu, H., and Chen, T. On the convergence rate of variants of the conjugate gradient algorithm in finite precision arithmetic. *SIAM Journal on Scientific Computing*, 43(5):496–515, 2021.

Hestenes, M. and Stiefel, E. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.

Higham, N. Analysis of the Cholesky decomposition of a semi-definite matrix. *Reliable Numerical Computation*, pp. 161–185, 1990.

Higham, N. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics (SIAM), 2nd edition, 2002.

Kiełbasiński, A. A note on rounding-error analysis of Cholesky factorization. *Linear Algebra and its Applications*, 88-89:487–494, 1987.

Korcyl, P. and Korcyl, G. Implementation of the conjugate gradient algorithm in lattice qcd on FPGA devices. *Proceedings of the 36th Annual International Symposium on Lattice Field Theory — PoS(LATTICE2018)*, 334:313, 2019.

Maddox, W., Potapcynski, A., and Wilson, A. Low precision arithmetic for fast Gaussian processes. In Cussens, J. and Zhang, K. (eds.), *The 38th Conference on Uncertainty in Artificial Intelligence*, pp. 1306–1316. Proceedings of Machine Learning Research, 2022.

Malakonakis, P., Isotton, G., Miliadis, P., Alverti, C., Theodoropoulos, D., Pnevmatikatos, D., Ioannou, A., Harteros, K., Georgopoulos, K., Papaefstathiou, I., and Mavroidis, I. Preconditioned conjugate gradient acceleration on FPGA-based platforms. *Electronics*, 11(19): 3039–3054, 2022.

Martin, S., Peters, G., and Wilkinson, J. Symmetric decomposition of a positive definite matrix. *Numerische Mathematik*, 7(5):362–383, 1965.

Matthews, A., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18 (40):1–6, 2017.

Meinguet, J. Refined error analyses of Cholesky factorization. *SIAM Journal on Numerical Analysis*, 20(6): 1243–1250, 1983.

Naher, J., Sakib, A., Jadhav, S., Gloster, C., and Doss, C. An FPGA based implementation of the conjugate gradient kernels. *International Conference on Electrical Information and Communication Technology (EICT)*, pp. 1–6, 2019.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

Pleiss, G., Gardner, J., Weinberger, K., and Wilson, A. Constant-time predictive distributions for Gaussian processes. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, pp. 4114–4123. Proceedings of Machine Learning Research, 2018.

Rasmussen, C. and Williams, C. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. The MIT Press, 2006.

Roldao, A. and Constantinides, G. A high throughput FPGA-based floating point conjugate gradient implementation for dense matrices. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 3(1):1–19, 2010.

Romano, J., Le, T., La Cava, W., Gregg, J., Goldberg, D., Chakraborty, P., Ray, N., Himmelstein, D., Fu, W., and Moore, J. PMLB v1.0: an open-source dataset collection for benchmarking machine learning methods. *Bioinformatics*, 38(3):878–880, 2021.

Shewchuk, J. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie-Mellon University, Pittsburgh, USA, 1994.

Song, L., Chi, Y., Guo, L., and Cong, J. Serpens: A high bandwidth memory based accelerator for general-purpose sparse matrix-vector multiplication. In Li, H. (ed.), *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 211–216. Association for Computing Machinery (ACM), 2022.

Song, L., Guo, L., Basalama, S., Chi, Y., Lucas, R. F., and Cong, J. Callipepla: Stream centric instruction set and mixed precision for accelerating conjugate gradient solver. In Zhang, Z. (ed.), *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 247–258. Association for Computing Machinery (ACM), 2023.

Sun, J.-g. Rounding-error and perturbation bounds for the Cholesky and ldlt factorizations. *Linear Algebra and its Applications*, 173:77–97, 1992.

Titsias, M. Variational learning of inducing variables in sparse Gaussian processes. In van Dyk, D. and Welling, M. (eds.), *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, pp. 567–574. Proceedings of Machine Learning Research, 2009.

Wang, K., Pleiss, G., Gardner, J., Tyree, S., Weinberger, K., and Wilson, A. Exact Gaussian processes on a million data points. *Advances in Neural Information Processing Systems*, 32, 2019.

Wilkinson, J. A priori error analysis of algebraic processes. In *Proceedings of the International Congress of Mathematicians*, pp. 629–.640, 1966.

Wilson, A. and Nickisch, H. Kernel interpolation for scalable structured Gaussian processes (kiss-gp). In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, pp. 1775–1784. Proceedings of Machine Learning Research, 2015.

Yang, D., Sun, J., Lee, J., Liang, G., Jenkins, D., Peterson, G., and Li, H. Performance comparison of Cholesky decomposition on GPUs and FPGAs. In Steffen, C. and Peterson, G. (eds.), *IEEE Symposium on Application Accelerators in High Performance Computing (SAAHPC'10)*. Proceedings IEEE Computer Society Press, 2010.

Appendix

A. Complete Experiment Results

The tables presented below exhibit the comprehensive results of all performed experiments.

Table 3. The impact of low-precision numeric representations on energy consumption and model performance for well-conditioned kernel matrices in Gaussian Process Regression with conjugate gradient implementation. Brackets in Precision column indicate the number of stable experiments to total experiments.

	Data	Precision	Conditioning	Δ Train	Δ Test	Δ UC	% Operations	Δ Energy
Conjugate Gradient	fr	3	10 ± 3	0.50	0.03	-0.14	97.99%	-98.38%
		4	11 ± 5	0.14	0.02	-0.06		-91.25%
		5	8 ± 1	0.07	0.01	-0.03		-94.63%
		6	9 ± 1	0.04	0.01	-0.01		-92.76%
		7	11 ± 3	0.02	0.03	-0.03		-90.88%
		8	9 ± 1	0.01	0.00	0.00		-89.01%
		14	10 ± 3	0.00	0.01	0.00		-75.98%
		24	9 ± 1	0.00	-0.01	0.01		-70.20%
		34	8 ± 1	0.00	0.01	-0.01		-40.10%
		44	8 ± 1	0.00	0.01	-0.01		-22.22%
		53	9 ± 1	-	-	-		-
	wn	3 (2/5)	80 ± 37	56.08	49.27	0.64	97.88%	-98.38%
		4	56 ± 7	0.42	0.07	-0.14		-91.24%
		5	60 ± 6	0.23	0.03	-0.08		-94.63%
		6	49 ± 9	0.12	0.02	-0.04		-92.75%
		7	56 ± 4	0.07	-0.02	-0.01		-90.88%
		8	65 ± 11	0.04	0.01	-0.02		-89.01%
		14	62 ± 10	0.00	0.01	0.01		-75.97%
		24	69 ± 22	0.00	0.03	-0.03		-70.21%
		34	67 ± 23	0.00	0.00	-0.01		-40.09%
		44	64 ± 16	0.00	0.01	-0.01		-22.22%
		53	79 ± 40	-	-	-		-
	st	4	96 ± 13	0.88	0.50	-0.28	97.27%	-91.21%
		5	127 ± 35	0.24	0.04	-0.10		-94.60%
		6	132 ± 42	0.12	0.02	-0.05		-92.73%
		7	126 ± 25	0.06	0.01	-0.02		-90.85%
		8	149 ± 36	0.02	0.01	-0.01		-88.98%
		14	89 ± 9	-0.01	0.02	-0.01		-75.92%
		24	128 ± 36	0.00	0.02	-0.02		-70.27%
		34	103 ± 11	-0.01	0.01	0.00		-40.04%
		44	117 ± 42	-0.01	0.01	-0.01		-22.23%
		53	152 ± 54	-	-	-		-
	mv	3 (4/5)	131 ± 26	1.53	0.89	-0.23	97.99%	-98.38%
		4	116 ± 48	0.17	0.01	-0.12		-91.25%
		5	116 ± 30	0.13	-0.01	-0.05		-94.63%
		6	131 ± 32	0.07	0.00	-0.02		-92.76%
		7	142 ± 58	0.04	0.00	0.00		-90.88%
		8	146 ± 63	0.02	0.01	-0.01		-89.01%
		14	111 ± 32	0.00	-0.01	0.01		-75.98%
		24	102 ± 23	0.00	0.00	0.00		-70.20%
		34	113 ± 22	0.00	0.01	0.00		-40.10%
		44	142 ± 126	0.00	0.01	0.00		-22.22%
		53	220 ± 190	-	-	-		-

Table 4. The impact of low-precision numeric representations on energy consumption and model performance for ill-conditioned kernel matrices in Gaussian Process Regression with conjugate gradient implementation. Brackets in Precision column indicate the number of stable experiments to total experiments.

	Data	Precision	Conditioning	Δ Train	Δ Test	Δ UC	% Operations	Δ Energy
Conjugate Gradient	pl	4 (4/5)	$6,758 \pm 2,368$	4.17	3.34	-0.07		-91.20%
		5 (4/5)	$12,428 \pm 10,904$	0.68	0.40	-0.17		-94.59%
		6	$12,242 \pm 7,638$	0.46	0.16	-0.18		-92.71%
		7	$23,176 \pm 31,356$	0.24	0.07	-0.11		-90.84%
		8	$8,274 \pm 5,364$	0.04	0.00	-0.03		-88.97%
		14	$13,214 \pm 7,143$	0.00	0.00	0.01	96.96%	-75.89%
		24	$14,396 \pm 13,138$	0.00	0.02	-0.01		-70.30%
		34	$14,203 \pm 10,190$	0.00	0.01	0.00		-40.02%
		44	$10,515 \pm 6,054$	0.00	-0.01	0.02		-22.23%
		53	$10,842 \pm 8,206$	-	-	-		-
	ch	4 (1/5)	$1,156,326 \pm 0$	144.37	144.82	0.59		-91.25%
		5 (4/5)	$3,483,705 \pm 1,927,678$	2.82	2.77	0.33		-94.64%
		6	$3,336,051 \pm 1,697,860$	1.38	1.29	0.10		-92.76%
		7	$5,423,509 \pm 3,604,730$	0.58	0.57	-0.06		-90.89%
		8	$3,845,724 \pm 1,986,413$	0.16	0.14	-0.25		-89.01%
		14	$3,394,682 \pm 2,659,590$	-0.03	-0.02	0.01	98.08%	-75.99%
		24 (4/5)	$3,647,790 \pm 2,357,889$	-0.03	0.00	-0.01		-70.19%
		34 (4/5)	$2,878,770 \pm 1,136,307$	-0.02	-0.01	0.00		-40.11%
		44	$2,526,487 \pm 1,514,300$	-0.03	0.01	-0.01		-22.22%
		53	$4,818,647 \pm 3,902,685$	-	-	-		-

Table 5. The impact of low-precision numeric representations on energy consumption and model performance for well-conditioned kernel matrices in Gaussian Process Regression with Cholesky decomposition implementation. Brackets in Precision column indicate the number of stable experiments to total experiments.

	Data	Precision	Conditioning	Δ Train	Δ Test	Δ UC	% Operations	Δ Energy
Cholesky Decomposition	fr	3 (4/5)	10 ± 2	0.12	0.04	-0.09	87.02%	-98.30%
		4	9 ± 1	0.10	0.02	-0.06		-91.16%
		5	12 ± 6	0.06	-0.02	0.01		-94.56%
		6	9 ± 1	0.04	0.00	-0.02		-92.68%
		7	10 ± 4	0.02	0.00	-0.01		-90.81%
		8	9 ± 1	0.01	-0.02	0.01		-88.94%
		14	9 ± 2	0.00	-0.02	0.01		-75.82%
		24	9 ± 1	0.00	0.00	0.00		-70.37%
		34	9 ± 2	0.00	-0.01	0.00		-39.96%
		44	8 ± 1	0.00	-0.01	0.01		-22.23%
		53	9 ± 2	-	-	-		-
	wn	3 (3/5)	57 ± 13	0.30	0.06	-0.15	86.52%	-98.26%
		4	59 ± 7	0.18	0.00	-0.07		-91.12%
		5	69 ± 16	0.11	0.01	-0.03		-94.52%
		6	61 ± 14	0.06	-0.03	0.01		-92.65%
		7	59 ± 13	0.03	-0.04	0.01		-90.77%
		8	54 ± 11	0.02	-0.02	0.01		-88.90%
		14	54 ± 7	0.00	-0.01	0.00		-75.74%
		24	62 ± 6	0.00	0.01	-0.01		-70.47%
		34	65 ± 12	0.00	0.00	-0.01		-39.89%
		44	59 ± 11	0.00	0.02	-0.03		-22.24%
		53	59 ± 9	-	-	-		-
	st	3 (4/5)	107 ± 7	0.27	0.07	-0.23	84.24%	-98.06%
		4 (4/5)	108 ± 16	0.15	0.03	-0.15		-90.91%
		5	103 ± 25	0.10	0.02	-0.08		-94.33%
		6	108 ± 22	0.05	0.01	-0.04		-92.46%
		7	101 ± 35	0.03	0.01	-0.02		-90.59%
		8	111 ± 24	0.02	-0.01	0.01		-88.73%
		14	114 ± 33	0.00	0.00	0.00		-75.34%
		24	114 ± 20	0.00	0.00	0.00		-70.90%
		34	112 ± 4	0.00	0.00	0.01		-39.54%
		44	117 ± 22	0.00	0.01	-0.01		-22.27%
		53	114 ± 20	-	-	-		-
	mv	4 (4/5)	135 ± 39	0.13	0.00	-0.12	87.02%	-91.16%
		5	142 ± 69	0.08	0.02	-0.07		-94.56%
		6	239 ± 190	0.05	0.00	-0.03		-92.68%
		7	187 ± 49	0.03	-0.01	-0.02		-90.81%
		8	192 ± 55	0.01	0.00	-0.01		-88.94%
		14	106 ± 16	0.00	0.00	-0.01		-75.82%
		24	130 ± 9	0.00	-0.01	0.00		-70.37%
		34	111 ± 23	0.00	0.00	0.00		-39.96%
		44	126 ± 82	0.00	0.00	-0.01		-22.23%
		53	127 ± 72	-	-	-		-

Table 6. The impact of low-precision numeric representations on energy consumption and model performance for ill-conditioned kernel matrices in Gaussian Process Regression with Cholesky decomposition implementation. Brackets in Precision column indicate the number of stable experiments to total experiments.

	Data	Precision	Conditioning	Δ Train	Δ Test	Δ UC	% Operations	Δ Energy
Cholesky Decomposition	pl	14	$25,093 \pm 39,926$	0.00	0.00	-0.01		-75.16%
		24 (4/5)	$66,100 \pm 107,969$	0.00	0.00	0.00		-71.09%
		34	$36,038 \pm 49,516$	0.00	0.00	0.00	83.27%	-39.38%
		44	$37,817 \pm 41,484$	0.00	0.00	0.01		-22.28%
		53 (4/5)	$12,815 \pm 10,215$	-	-	-		-
	ch	24	$4,574,294 \pm 2,490,886$	0.00	0.05	-0.01		-70.30%
		34	$2,856,070 \pm 1,929,682$	0.00	0.01	-0.03		-40.02%
		44	$3,802,740 \pm 1,775,298$	0.00	-0.02	-0.03	87.43%	-22.23%
		53	$3,222,575 \pm 1,872,118$	-	-	-		-

B. Results of the Experiments With Larger Exponents

The results of the experiments in the following table explored the impact of increasing the number of exponent bits from 11 to 15 on achieving stable results. Our investigation focused on discerning whether the precision or absolute range of the exponent was more crucial for stability. Precision levels below the minimum precision continued to show the same instability: 4 and 5 bits for the 'ch' dataset and 3 to 8 bits for the 'pl' dataset. Precisions above the minimum precision benefited from the larger exponents with stable results at 24 and 53 bits for the 'pl' dataset using Cholesky decomposition, and at 34 bits for the 'ch' dataset using the conjugate gradient method. However, 24 bits with the conjugate gradient on the 'ch' dataset remained unstable. These patterns of stability and instability were reproducible in repeated trials with random draws.

Table 7. The impact of using low-precision numeric representations with exponent bits set to 15 instead of 11. Experiments were performed on the ill-conditioned kernel matrices from the 'pl' and 'ch' datasets for previously unstable experiment settings.

Approach	Data	Precision	Conditioning	Δ Train	Δ Test	Δ UC	% Operations	Δ Energy
Conjugate Gradient	ch	4 (1/5)	$1,863,689 \pm 0$	8.10	7.07	0.49	98.08%	−91.25%
		5 (4/5)	$3,229,871 \pm 2,869,910$	7.15	6.80	0.42		−94.64%
		24 (4/5)	$2,005,146 \pm 1,410,845$	−0.01	0.01	−0.01		−70.19%
		34	$2,957,468 \pm 1,974,483$	0.01	0.03	−0.02		−40.11%
Cholesky Decomposition	pl	24	$16,288 \pm 9,853$	0.00	−0.01	0.01	83.27%	−71.09%
		53	$11,316 \pm 5,253$	-	-	-		-

C. Algorithms for Efficient Kernel Matrix Inversion

The inverse of a kernel matrix can be expressed as a system of linear equations $K_{X,X} \cdot [K_{X,X}]^{-1} = I$ which can be solved to obtain the inverse. Nevertheless, implementations of Gaussian process regression (GPR) do not directly solve for the inverse but rather for the more efficient matrix product of the inverse with the target variable. Using the kernel matrix $K_{X,X}$ and the target variable vector $\mathbf{y}$, we can formulate the system of linear equations $K_{X,X} \cdot \mathbf{s} = \mathbf{y}$.

When rearranging the term to $\mathbf{s} = [K_{X,X}]^{-1}\mathbf{y}$, we obtain the right side of the predictive mean from Equation (2) by the solution vector s. Similarly, this approach can be applied to formulate a system $S = [K_{X,X}]^{-1}K_{X,X_*}$ to obtain the right sight of Equation (3) of the predictive covariance. In contrast to the system of the predictive mean, the solution S of the system of the predictive covariance is a matrix consisting of as many solution vectors $\mathbf{s_i}$, as test points exist in X_*.

C.1. Cholesky Decomposition

Those systems can be efficiently solved if decomposing $K_{X,X}$ in a triangular system, as we can then leverage forward and backward substitution to obtain the solution vectors.

The Cholesky decomposition is the most popular method for inverting kernel matrices in GPR. It takes advantage of the symmetry and positive definiteness of the kernel matrix, thus reducing the number of arithmetic operations required. The Cholesky decomposition is a type of LU factorization, where $U = L^T$. It decomposes the kernel matrix $K_{X,X}$ into a lower triangular matrix L and an upper triangular matrix L^T, such that $L \cdot L^T = K_{X,X}$. This process requires $O(\frac{1}{6}n^3)$ complexity. In total, we obtain $O(\frac{1}{6}n^3)$ for the Cholesky decomposition and $O(n^2)$ for the forward and backward substitutions, the latter being applied each one time for the predictive mean (independent of the number in test points). However, for the predictive covariance, we apply forward and backward substitutions each time for any test point. Forward and backward substitution are dominated by multiplications and additions.

The Cholesky decomposition is a fundamental element of any real-world GPR implementation and the common default implementation. Alternative methods to the Cholesky decomposition can be considered, yet they are not commonly used. For instance, the LDL decomposition can be viewed as an extension of the Cholesky decomposition and eliminates costly and potentially numerically unreliable square-root operations. However, it adds overhead by introducing a unit diagonal into the factorization (Bunch & Kaufman, 1977). The Bunch-Kaufman factorization builds on the LDL decomposition by adding a permutation matrix to increase numerical stability (and adds further overhead) (Bunch & Kaufman, 1977). The Singular Value Decomposition (SVD) can invert the kernel matrix if it is nearly singular. However, it is not as memory and runtime efficient as the Cholesky decomposition. The Cholesky decomposition is the most efficient method for GPR, and the kernel matrix is usually considered sufficiently conditioned, so these alternatives are rarely used. The Cholesky decomposition is known to be a numerically stable process, and in the case of infinite arithmetic, the kernel matrix always has a Cholesky decomposition (Golub & Loan, 2013). The exploration of these alternatives could serve as a direction for future research, aiming to further reduce the required minimum precision for the Cholesky decomposition.

Algorithm 1 presents the implementation of GPR using the Cholesky decomposition according to Rasmussen & Williams (2006). The Cholesky-Banachiewicz algorithm (Banachiewicz, 1938) is used on line 1 to build the lower triangular matrix row by row. Each lower triangular element is derived by first taking the equivalent element from the original matrix. If it is a diagonal element, we subtract the sum of the squared elements in the same row of L left to the diagonal element before taking the square root of the term. If it is not a diagonal, we subtract the sum of the products of each element of the same row to the left of L and the element of the same column from the above row. We divide the term by the diagonal element of the same column. Intuitively, this process is sometimes compared to completing the square for polynomials or calculating the square root of a matrix. The vast amount of arithmetic operations consists of the multiplications and additions in line 13 and is dependent on the size of the kernel matrix. Other implementations, such as the Cholesky-Crout algorithm (Crout, 1941), take a column-by-column approach, which does not change the required number (or type) of arithmetic operations.

Common high-level library implementations for GPR (e.g., scikit-learn) use a block version (Golub & Loan, 2013) of the Cholesky decomposition that is highly efficiently implemented in LAPACK (DPOTRF), also leveraging BLAS operations. This introduces computational overhead but leverages modern hardware architecture for efficiency gains through parallelization. Independent of the respective implementation or extension, the overall complexity of the algorithm is always the same. For operation counting, we considered the Cholesky-Banachiewicz algorithm as a representative and well-analyzable implementation of the Cholesky decomposition without any unnecessary overhead. Insights regarding stability, performance, and energy savings with low-precision arithmetic translate directly into the widely used block version.

Algorithm 1 Gaussian Process Regression with the Cholesky–Banachiewicz Decomposition (Rasmussen & Williams, 2006)

Require: Training features X, training target vector $\mathbf{y}$, (one) test feature x_*

1: $L \leftarrow \text{cholesky}(K_{X,X})$
2: $\boldsymbol{\alpha} \leftarrow L^T \backslash (L \backslash \mathbf{y}))$
3: $\mathbf{v} \leftarrow L \backslash K_{x_*,X}$
4: $\bar{f}_* \leftarrow K_{x_*,X}^T \boldsymbol{\alpha}$
5: $\text{cov}[\bar{f}_*] \leftarrow K_{x_*,x_*} - \mathbf{v}^T \mathbf{v}$
6: **function** cholesky(K)
7: $L \leftarrow \mathbf{0}$ # initialize L as zero matrix of same dimensions as K
8: $n \leftarrow \text{rows}(K)$ # number of rows in K
9: **for** $i = 0$ to n **do**
10: **for** $k = 0$ to i **do**
11: $sum \leftarrow 0$
12: **for** $j = 0$ to $k - 1$ **do**
13: sum $\leftarrow$ sum $+ L[i][j] \cdot L[k][j]$
14: **end for**
15: **if** $i = k$ **then**
16: $L[i][k] \leftarrow \sqrt{K[i][i] - \text{sum}}$
17: **else**
18: $L[i][k] \leftarrow \frac{1}{L[k][k]}(K[i][k] - \text{sum})$
19: **end if**
20: **end for**
21: **end for**
22: **return** L

We apply the forward and backward substitutions ('\') in line 2 for the predictive mean, while the predictive variance in line 3 needs forward substitution for every test point. Algorithm 1 only utilizes inference for a single test point x_* for simplicity. We have written x_* as a scalar in the pseudocode, implying an isotropic RBF kernel. Finally, we use $\boldsymbol{\alpha}$ and $\mathbf{v}$ to obtain the predictive mean in line 4 and predictive covariance in line 5. Algorithm 1 illustrates that multiplications, additions, and subtractions are the majority of arithmetic operations in the Cholesky decomposition, followed by divisions and the square root operation.

While the influence of round-off error due to low-precision floating-point representations for GPR is empirically evaluated in this work, theoretical round-off error bound analysis for the Cholesky decomposition with floating-point arithmetic were derived (Martin et al., 1965; Wilkinson, 1966), sharpened, and extended by several authors (Meinguet, 1983; Kiełbasiński, 1987; Higham, 1990; Sun, 1992; Higham, 2002). While they do not tell the influence of round-off error on GPR performance and only give worst-case bounds, they give hints about what contributes to GPR quality eventually.

Martin et al. (1965) bound the additive error matrix F in $LL^T = K + F$ by its spectral norm $||F||_2$. They bound the error by

$$||F||_2 \leq \epsilon \cdot n^{3/2} \cdot 2^{-t} \cdot ||K||_2, \tag{8}$$

where t is the number of mantissa bits, n is the rank of K, and ϵ is an error factor induced by rounding. They assume the error ϵ to be around unity when using Round Half To Even, hence not being a significant overall error source. The most important implication from this upper error bound is that the Cholesky decomposition might be robust to round-off error. Furthermore, not only does the precision (2^{-t}) have an impact on the error but also the conditioning of the matrix K and the size of the kernel matrix (hence the size of our training data).

C.2. Conjugate Gradient

The Cholesky decomposition focuses on solving the introduced systems of linear equations by decomposing the kernel matrix into a triangular system and applying forward and backward substitutions. The conjugate gradients approach focuses on iteratively solving the same systems of linear equations.

The conjugate gradient algorithm can be used to precisely solve the linear system of equations. The precise solution can be

Algorithm 2 Conjugate Gradient for Gaussian Process Regression (Hestenes & Stiefel, 1952; Davies, 2005; Maddox et al., 2022)

Input: Kernel matrix K, right hand side vector $\mathbf{v}$, maximum iterations i
Initialize: $k \leftarrow 0$, $\mathbf{x}_0 \leftarrow 0$, $\mathbf{r}_0 \leftarrow K\mathbf{x}_0 - \mathbf{v}$, $\mathbf{d}_0 \leftarrow -\mathbf{r}_0$
while $k < i$ **do**
  $\alpha_k \leftarrow \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{d}_k^T K \mathbf{d}_k}$
  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{d}_k$
  $\mathbf{r}_{k+1} \leftarrow \mathbf{r}_k + \alpha_k K \mathbf{d}_k$
  $\beta_{k+1} \leftarrow \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$
  $\mathbf{d}_{k+1} \leftarrow -\mathbf{r}_{k+1} + \beta_{k+1} \mathbf{d}_k$
  $k \leftarrow k + 1$
end while

achieved by conducting iterations equal to the rank of the kernel matrix, which corresponds to the number of training data observations. While the approach has the same computational overall complexity of $O(n^3)$ as the Cholesky decomposition, it is computationally more expensive in absolute terms.

A conjugate gradient approach for GPR was suggested by Gibbs and McKay (1997) and was further developed by Davies (Davies, 2005). It has been implemented in libraries such as GPytorch (Gardner et al., 2018). When using the conjugate gradient in the context of GPR, it is generally assumed that a satisfactory approximation of the solution to the linear system can be obtained in fewer than the full number of iterations. Conjugate gradient reduces the complexity of inversion to $O(n^2 \cdot i)$ if we terminate the conjugate gradient algorithm after i iterations. Reducing the iterations helps to reduce runtime and balance performance with the computing power and energy used. However, the same number of iterations will not always produce the same performance for different kernel matrices. The conditioning of the matrix is a key factor in determining how many iterations are needed to achieve a certain level of approximation, as the number of iterations is proportional to the square root of the condition number (Davies, 2005).

Algorithm 2 presents the vanilla conjugate gradient approach. The conjugate gradient algorithm comprises the solution vector $\mathbf{x}$, the residual vector $\mathbf{r}$, and the search directions $\mathbf{d}$ as its fundamental components. The current iteration is tracked by variable k and checked against the maximum number of iterations specified in i. After finishing the desired number of iterations, the solution is in vector $\mathbf{x}$. The generic right-hand side $\mathbf{v}$ is occupied by $\mathbf{y}$ for the predictive mean or by the appropriate vectors for the predictive covariance.

We typically start with $\mathbf{x}$ being the zero vector and incorporate the residual vector $\mathbf{r}$ into our search direction $\mathbf{d}$. Each of these vectors is modified in each iteration of the algorithm. The vector $\mathbf{x}$ is updated according to the search direction $\mathbf{d}$ and the scaling factor α, which is the optimal step size. The scalar α includes information about the magnitude of the current residual error (how far do we have to go to reduce the error) and about the alignment of the current search direction with the function curvature of matrix K applied on $\mathbf{d}$ (how does the shape in K affect the change in error for the search direction).

The residual is adjusted based on its previous value, the step size, and the search direction altered by the matrix K. The search direction is then modified based on the new residual and the previous search direction, with β applied. The term conjugate in conjugate gradient is derived from the fact that a search direction is always K-orthogonal to prior search directions. A search direction $\mathbf{d}$ is K-orthogonal for a matrix K if $\mathbf{d}_i^T K \mathbf{d_j} = 0$ (Shewchuk, 1994). The scalar β guarantees that the new search direction is K-orthogonal to the previous search directions.

While it is known that a smaller condition number leads to faster conjugate gradients performance, Greenbaum et al. (2021) investigated the convergence characteristics of various conjugate gradients implementations when using finite precision arithmetic (meaning double-precision versus infinite precision). The authors demonstrate that the spread of eigenvalues is the primary element that influences the speed of exact conjugate gradient convergence, and it determines how quickly conjugate gradients converge when using finite precision. In addition to the vanilla conjugate gradients implementation of Hestenes & Stiefel (1952), which was discussed in Algorithm 2, they examine the conjugate gradient algorithm by Chronopoulos & Gear (1989) for improved parallelization and a pipelined version of Ghysels & Vanroose (2014). The classical conjugate gradients implementation needed the least or a similar amount of iterations for the test problems (which differed in their conditioning).

Algorithm 3 Modified Conjugate Gradient for Gaussian Process Regression (Maddox et al., 2022)

1: **Input:** Kernel matrix K, right hand side vector $\mathbf{v}$, iterations i, preconditioner matrix P^{-1}
2: **Initialize:** $k \leftarrow 0$, $\mathbf{x}_0 \leftarrow \mathbf{0}$, $\mathbf{r}_0 \leftarrow K\mathbf{x}_0 - \mathbf{v}$, $\mathbf{d}_0 \leftarrow \mathbf{r}_0$, $\mathbf{z}_0 \leftarrow P^{-1}\mathbf{r}_0$, $\log(\gamma_0) \leftarrow \text{logSumExp}(\mathbf{r}_0^T\mathbf{z}_0)$
3: **while** $k < i$ **do**
4: $\quad\alpha_k \leftarrow \exp(\log(\gamma_k) - \text{logSumExp}(\mathbf{d}_k^T K\mathbf{d}_k))$
5: $\quad\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k\mathbf{d}_k$
6: $\quad\mathbf{r}_{k+1} \leftarrow \mathbf{r}_k + \alpha_k K\mathbf{d}_k$
7: $\quad$**for** $j = 0$ to k **do**
8: $\quad\quad\mathbf{r}_{k+1} \leftarrow \mathbf{r}_{k+1} - (\mathbf{u}_j^T\mathbf{r}_{k+1})\mathbf{u}_j$
9: $\quad$**end for**
10: $\quad\mathbf{z}_{k+1} \leftarrow P^{-1}\mathbf{r}_{k+1}$
11: $\quad\log(\gamma_{k+1}) \leftarrow \text{logSumExp}(\mathbf{r}_{k+1}^T\mathbf{z}_{k+1})$
12: $\quad\beta_{k+1} \leftarrow \exp(\log(\gamma_{k+1}) - \log(\gamma_k))$
13: $\quad\mathbf{d}_{k+1} \leftarrow -\mathbf{r}_{k+1} + \beta_{k+1}\mathbf{d}_k$
14: $\quad\mathbf{u}_{k+1} \leftarrow \mathbf{d}_{k+1}$
15: $\quad k \leftarrow k + 1$
16: **end while**

Greenbaum et al. defined machine error for a matrix as a perturbed matrix whose eigenvalues lie within certain intervals around the original eigenvalues. Greenbaum (1989) had previously shown that, under certain assumptions, the convergence behavior of conjugate gradients of a perturbed matrix is similar to that of a matrix whose eigenvalues are located in small intervals around the eigenvalues of the original matrix. The eigenvalue distribution and the size of the intervals determined by the machine error can prolong the convergence of the conjugate gradients method compared to the original matrix.

Greenbaum et al. (2021) formalized theoretical upper bounds δ for the residual vector $\mathbf{r_k}$ and the search direction vector $\mathbf{d_k}$ for one iteration step that we can use to determine important factors for our work with (low) finite precision. The spectral norm for a matrix and the L2 norm for a vector is denoted by $||\cdot||$, c is a constant, and ϵ is the maximum relative error. The upper bounds for the residual vector $||\delta_{\mathbf{r_k}}||$ and the search direction $||\delta_{\mathbf{d_k}}||$ are

$$||\delta_{\mathbf{r_k}}|| \leq \epsilon \cdot (||\mathbf{r_{k-1}}|| + 2||\alpha_{k-1}K\mathbf{d_{k-1}}|| + c||K||\,||\alpha_{k-1}\mathbf{d_{k-1}}||) + O(\epsilon^2) \tag{9}$$

and

$$||\delta_{\mathbf{d_k}}|| \leq \epsilon \cdot (||\mathbf{r_k}|| + 2||\beta_k\mathbf{d_{k-1}}||) + O(\epsilon^2). \tag{10}$$

In addition to the error caused by the floating-point precision, we observe that the conditioning of K or its related terms plays an important role in the bounds. The spectral norms either contain K or implicitly depend on K. The value of a matrix's conditioning number is directly related to its spectral norm. Thus, poor conditioning leads to higher upper error bounds. The conditioning of the other conjugate gradients terms in the error bounds eventually depends on K. Overall, the implications are similar to the upper bounds for the Cholesky decomposition. Besides the round-off error ϵ, the conditioning significantly influences the eventual error. Furthermore, both scale each other by multiplication. Finding the right floating-point representation for practical applications is not straightforward from those bounds, necessitating empirical trials. The upper limits, however, give us useful information.

The conjugate gradients approach exhibits matrix-vector operations (multiplications and additions) as dominating operations dependent on the kernel matrix size and number of performed iterations.

Maddox et al. (2022) introduced the modified conjugate gradients algorithm presented in Algorithm 3, which we employed. They used matrix-vector operations in half-precision to process larger datasets, allowing bigger kernel matrices. They employed several measures to improve the robustness of GPR during half-precision computation. To prevent round-off errors and overflows, they implemented three modifications.

They decreased the chance of exponent overflows by rescaling the kernel matrix-vector multiplications. The vector is downscaled by factor $\frac{1}{\sqrt{n}}$, where n is the rank of K. This has an impact on the calculations of the residual r_{k+1} on line 6 and the alpha value α_k on line 4 (as well as their initial values).

Second, they store and calculate α and β (lines 4, 11-12, and initialization) using a logarithmic scale and then convert it back to linear scale for their respective applications in the solution, residual and search direction vectors. Storing in the logarithmic scale downscales large values. It upscales subnormal values, thus reducing over- and underflows and improving the round-off error of calculations.

Maddox et al. (2022) calculate the product of two vectors $\mathbf{a}$ and $\mathbf{b}$ using the LogSumExp function by computing $\text{logSumExp}(\mathbf{a}^T \cdot \mathbf{b}) = y_{max} + \log(\sum_{i=1}^{n} s_i \cdot \exp(\mathbf{y_i} - y_{max}))$, where $y_i = \log(a_i) + \log(b_i)$ and $s_i = \text{sign}(a_i \cdot b_i)$ with a_i and b_i denoting the vector elements. On the one hand, y_{max} downscales the products of the individual vector elements by the largest product, thus increasing the accuracy and preventing large values from being lost in the final sum. The addition of y_{max} to the full terms scales it back.

To illustrate the differences, consider the term $\frac{p \cdot z}{q} = \exp(\log(p) + \log(z)) - \log(q)$ with $p = z = q = 1 \times 10^{-250}$. While in mathematics with infinite precision, these terms are equivalent, resulting in a value of 1×10^{-250}, double-precision causes the nonlogarithmic representation to be 0, as the term $p \cdot z$ causes an underflow.

As the third measure, they employ Gram-Schmidt re-orthogonalization in each iteration (lines 7-8) to maintain the orthogonality of the residual vectors to any previous residual vector $\mathbf{u_i}$ that may have been lost due to the round-off errors from the kernel matrix multiplications, as suggested by Gratton et al. (2021).

The condition number of the kernel matrix largely determines the number of iterations needed for convergence. To decrease the condition number and thus the speed of convergence, they used a preconditioner P^{-1}. In our experiments, we did not include the preconditioner application.