Electronic Supplementary Material (ESI) for Digital Discovery.

Electronic Supporting Information (ESI)

FEREBUS: A High-Performance Modern Gaussian Process Regression Engine

Matthew Burn and Paul Popelier

Department of Chemistry, The University of Manchester, Manchester, M13 9PL, (Great) Britain

*To whom correspondence should be addressed:

Phone: +44 161 3064511. E-mail: pla@manchester.ac.uk

Table of Contents

1 FEREBUS Benchmarks

1.1 Hardware

Table S1. CPU Node Architecture used.

Architecture	Skylake
CPU	2×16-core Intel Xeon Gold 6130 @ 2.10GHz
Memory	192Gb RAM

Table S2. GPU Node Architecture used.

Architecture	Skylake
GPU	Nvidia v100-SXM2-16GB (Volta)
CPU	2×16-core Intel Xeon Gold 6130 @ 2.10GHz
Memory	192Gb RAM

1.2 Optimisation Parameters

Table S3. PSO Parameters used.

Parameter	Value
Iterations	1000
Swarm Size	64
Inertia Weight	0.729
Cognitive Learning Rate	1.494
Social Learning Rate	1.494
Stopping Criterion	No Stopping Criteria

Table S4. Model Parameters used.

Number of Training Points	2130
Number of Dimensions	51
Mean	Constant
Kernel	RBF

1.3 Timings

Table S5. CPU benchmark timings.

$n_{threads}$	**Time Taken**	**Parallel Efficiency**
1	13:02:10	
2	06:37:04	98%
4	03:18:14	99%
8	01:40:18	97%
16	00:51:14	95%
32	00:27:20	89%

Table S6. GPU benchmark timings.

$n_{threads}$	**Time Taken**	**Parallel Efficiency**
1	02:52:05	
2	01:31:49	94%
4	00:44:51	96%
8	00:29:17	73%

Table S7. Comparison between CPU and GPU benchmarks.

$n_{threads}$	**CPU Time Taken**	**GPU Time Taken**	**Speedup Factor**
1	13:02:10	02:52:05	4.55
2	06:37:04	01:31:49	4.32
4	03:18:14	00:44:51	4.42
8	01:40:18	00:29:17	3.43

2 Kernel Interpreter

In FEREBUS, a composite consists of multiple kernels combined through the addition and multiplication operations. Adding and multiplying kernels is as simple as implementing two composite kernel types: k_{sum} and k_{prod} for adding and multiplying kernels, respectively:

$$k_{sum}(x, x^*) = k_1(x, x^*) + k_2(x, x^*) \quad \text{(S1)}$$

$$k_{prod}(x, x^*) = k_1(x, x^*) \times k_2(x, x^*) \quad \text{(S2)}$$

Interpreting the composite kernel written as a string of characters in a config file (such as '`k1*k2`') involves the use of a kernel interpreter. A kernel interpreter is a subroutine designed to take a string of characters and return a kernel to be used by the GPR model. The method for generating the composite kernel involves lexing the input string into tokens, parsing these tokens into an *abstract syntax tree* (AST), and then interpreting the AST into a composite kernel.

The first step (lexing) uses a lexical analyser (also known as lexer) to scan the input characters separating the characters into tokens. A token is a single unit of an expression, which can be a number, a variable name, an operator and so on. A full list of tokens is shown in the Table **S8**. If the lexer comes across an unknown token, it will exit with an error.

Table S8. All token types used within the FEREBUS kernel lexer.

Token Name	Example
Number	`1, 2.0, 1e-3, etc.`
Add Operator	`+`
Subtract Operator	`-`
Multiply Operator	`*`
Divide Operator	`/`
Left Pathenthesis	`(`
Right Parenthesis	`)`
Variable Name	`k1, k2, myvar1, myvar2, etc.`
End Of File	

Once the input string has been lexed into tokens they must be parsed into an AST to allow the interpreter to determine the order in which the tokens must be evaluated. The parser forms expressions within a tree that can be evaluated, a single node at a time, in order to return the desired result. The parser is not only responsible for identifying expressions from the tokens but also for preserving the correct mathematical order of operations (including parentheses). The parser is implemented using the well-known recursive descent parsing algorithm. Figure **S1** shows a schematic for an example interpreter workflow. It is the parser's responsibility to determine the correctness of the expression. If the expression is incorrect (for example, if a parenthesis is missing), the parser will exit with an error.

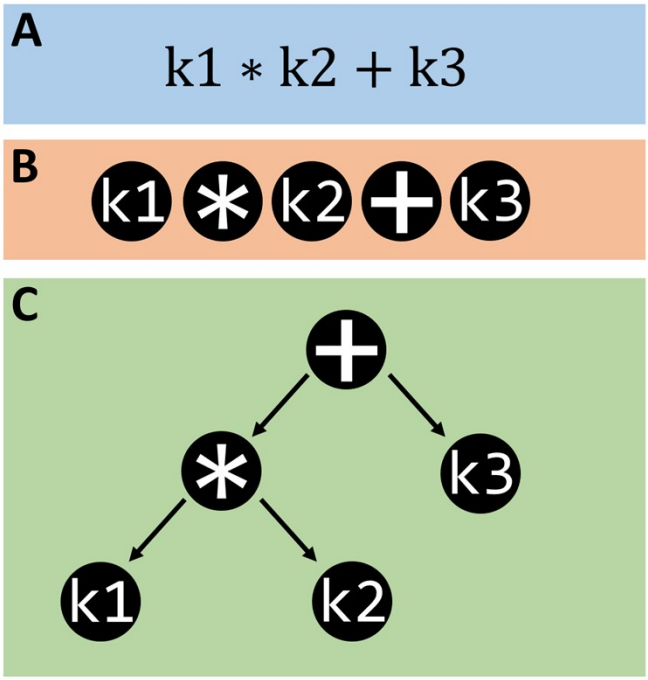

Figure S1. Schematic for interpreting an example composite kernel: (a) composite kernel input string, (b) lexed tokens, and (c) AST produced by the recursive descent parser.

The kernel interpreter then walks through the AST (which is always read from top to bottom) from the head node (the first node, in the case of Figure S1, the '+' operation node) substituting any variable names with the kernel matching the variable name defined in the config file. The kernel interpreter works within a namespace defined by the config file. If a variable name appears that does not correspond to a kernel defined in the FEREBUS config file, the interpreter will exit with an error. Once the AST has been successfully walked through, a composite kernel is returned, to be used by the GPR model for hyperparameter optimisation.

3 Likelihood Functions

The optimisation of the GPR model's hyperparameters requires a cost function in order to provide a best fit to the training data. The standard cost function used for Gaussian processes is the marginal likelihood function denoted LL. For a given training set (training inputs, X, and training outputs y) and hyperparameters (θ), the marginal log-likelihood is defined as follows,

$$LL(y \mid X,\theta) = -\frac{1}{2}(y-\mu)^{\top}R^{-1}(y-\mu) - \frac{1}{2}ln|R| - \frac{n}{2}ln2\pi \tag{S3}$$

By setting the derivative of the log-likelihood with respect to the mean to zero, the mean is optimised analytically such as to produce the concentrated mean, $\hat{\mu}$, or

$$\frac{dLL}{d\mu} = 0 \Rightarrow \hat{\mu} = \frac{1^{\top}R^{-1}y}{1^{\top}R^{-1}1} \tag{S4}$$

where 1 denotes a vector of ones in the same shape as the training output vector, y. The concentrated mean may then be used to calculate the concentrated variance,

$$\hat{\sigma}^2 = (y-\hat{\mu})^{\top}R^{-1}(y-\hat{\mu}) \tag{S5}$$

The concentrated variance may then be substituted into the marginal log-likelihood equation (S3) to produce the concentrated log-likelihood ($\hat{LL}$),

$$\hat{LL}(y \mid X,\theta) = -\frac{1}{2}\hat{\sigma}^2 - \frac{1}{2}ln|R| - \frac{n}{2}ln2\pi \tag{S6}$$

FEREBUS implements both likelihood functions thereby allowing the user to select the likelihood function based upon their use case with the default likelihood function being the marginal log-likelihood. In practice we observed that the concentrated likelihood function produces a more accurate model but takes more PSO iterations to optimise. The difference in predictive accuracy between the two likelihood functions depends on the system being modelled as shown by the plots in Figures S2 and S3. Thus, providing the option to select the appropriate likelihood function to the user is necessary.

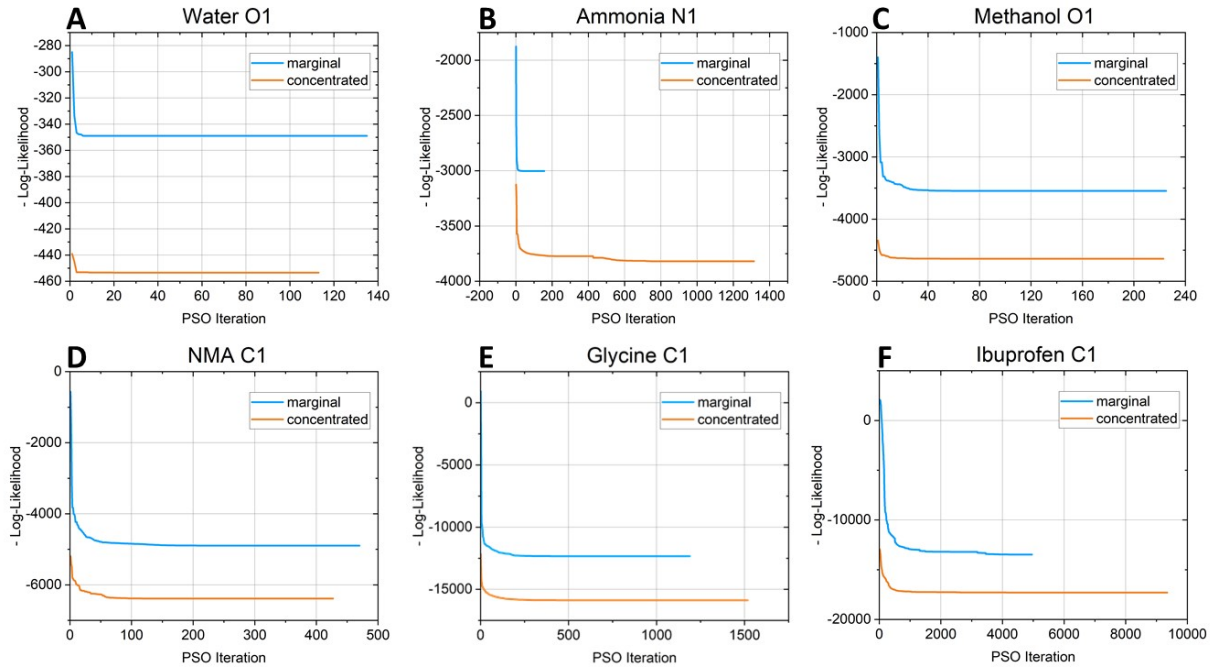

Figure S2. Plots showing the likelihood value *versus* the PSO iteration for several systems.

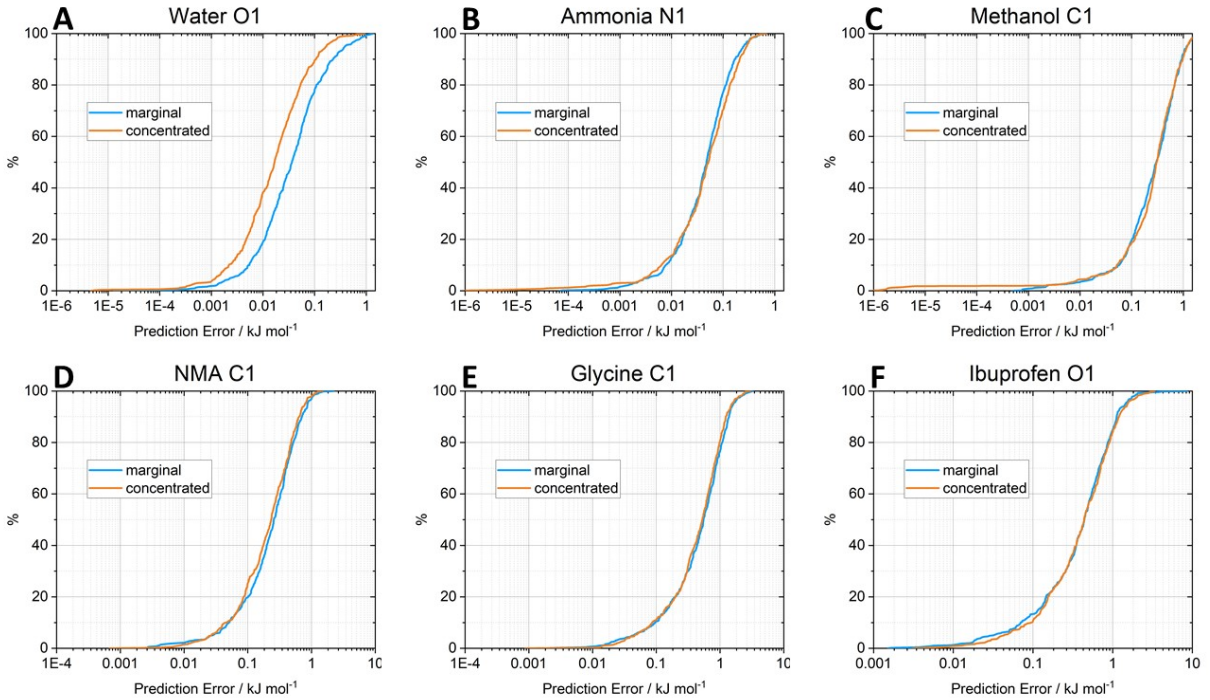

Figure S3. Plots showing the prediction errors from the models resulting from the optimisations shown in Figure S2.

Table S9. Table showing the model details, RMSE and maximum prediction errors (kJ mol^{-1}) for the atomic energies in each model shown in Figures S2 and S3.

System Name	$n_{features}$	n_{train}	Marginal		Concentrated	
			RMSE	Max Error	RMSE	Max Error
Water	3	59	0.20	1.34	0.10	0.94
Ammonia	6	421	0.11	0.58	0.13	0.69
Methanol	12	636	0.58	3.66	0.56	2.07
N-methylacetaimde (NMA)	30	887	0.43	2.23	0.39	1.50
Glycine	51	2130	0.86	3.24	0.80	2.98
Ibuprofen	93	2264	0.83	8.59	0.78	3.33