The Art of Inhibiting Patterns: Gaussian Process Optimization of Hyphenation

Anonymous ACL submission

Abstract

The quality of hyphenation matters. It does matter when rendering texts on small mobile screens, when rendering HTML in any browser, and it matters in the narrow columns of academic papers, such as in this abstract. Having an automated, robust, efficient, and generalizable solution for all languages in use would have saved billions of hours of scrolling and millions of tons of paper.

The half-century of attempts to solve the hyphenation problem yielded usable solutions with TEX hyphenation patterns, but with low coverage and errors caused mainly by insufficient exception handling, compound words, and language changes (including new words, transcriptions). The setting of parameters of the pattern generation process is still done by trial and error.

We have designed a new method of Gaussian process optimization of hyphenation pattern generation by the `patgen` program. It optimizes threshold setting for coverage and inhibition levels of pattern generation. We utilized a collection of 18 diverse datasets comprising a hyphenated word list for 15 languages and generated nearly optimal hyphenation patterns with respect to precision, recall, and generalization (10-fold cross-validation) optimization criteria. The $F_{1/7}$ score reaches 0.9992 for consistently marked word lists, such as Czech and Slovak.

A newly designed optimization technique for the pattern generation methodology has set new quality standards, approaching an optimal solution. Its deployment in typesetting systems like TEX, InDesign, or in text processors, as well as in CSS hyphenation in web browsers, would yield a huge impact.

"Perfection is not attainable, but if we chase perfection, we can catch excellence." Vince Lombardi

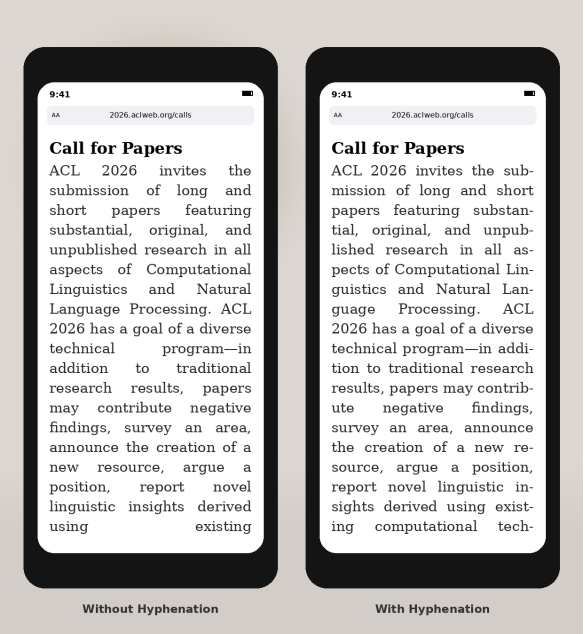


Figure 1: Comparison of text rendering on a mobile device. Note the 'rivers' of whitespace without hyphenation (left) and the improved spacing density with hyphenation and fewer lines for scrolling (right).

1 Motivation

The quality of hyphenation matters. It matters when rendering HTML texts on small mobile screens, as shown in Figure 1. It matters in the narrow columns of academic papers, such as in the two-column papers in ACL Anthology, as in this paragraph.

Having an automated, robust, efficient, and generalizable solution for all languages in use would have saved billions of hours of scrolling and millions of tons of paper.

2 The Development of Hyphenation Algorithms

Approaches have evolved from pattern-based and rule-based systems to various statistical machine learning methods, including neural.

Pattern-Based Hyphenation

The digital pursuit of hyphenation quality began in 1983 when Donald Knuth's doctoral student Frank Liang (Liang, 1983) designed a supervised learning method to generate hyphenation ***patterns*** from a list of already hyphenated words of Webster's Pocket Dictionary, for use in TEX (Knuth, 1986, Appendix H). The algorithm implemented in the `patgen` program (Liang and Breitenlohner, 2020) employs a dynamic packed trie data structure to efficiently store patterns and identify hyphenation points based on local ***context*** windows. It operates on a multilevel system where patterns can either enable or inhibit the hyphenation point, based on weighted values. This method is renowned for its exceptional space efficiency and high coverage. Liang's `hyphen.tex` with 4,447 patterns covers 89.3% of hyphens in the input word list.

Sojka (1995) showed that the patterns could be generated with coverage of 98.37% for German, and that the method could be used for segmentation of compounds, too.

Finite State Methods Bouma (2003) explores the formalization of hyphenation using Finite State Transducers (FSTs). Unlike the specialized packed trie implementation found in TEX, this approach utilizes standard finite state calculus, where hyphenation patterns and rules are expressed as regular expressions. This method demonstrates that hyphenation can be effectively modeled using deterministic automata, providing a rigorous mathematical framework that allows the seamless composition of hyphenation with other finite-state morphological processes while maintaining accuracy comparable (99.1% accuracy) to traditional pattern-based systems (99.8% accuracy) on running Dutch text (Bouma, 2003, Table 5).

Subword Segmentation by Transformers While distinct from typographic hyphenation (which focuses on line breaking), subword segmentation is the dominant paradigm in modern NLP for training Large Language Models (LLM). Sennrich et al. (Sennrich et al., 2016) introduced Byte-Pair Encoding (BPE) for neural machine translation, which iteratively merges frequent character pairs to form subword units.

Language-Agnostic Neural Syllabification A language-agnostic approach, framing syllabification as a sequence labeling task utilizing Bidirectional Long Short-Term Memory (Bi-LSTM) networks, has been proposed by Krantz et al. (2019). By treating hyphenation points as classification targets between characters, they demonstrate that high-accuracy segmentation can be achieved directly from raw text data without reliance on language-specific rules or manual feature engineering. This contrasts with the `patgen` approach by prioritizing generalization capabilities through deep learning over the storage efficiency of packed tries. Franek (Franěk, 2025) measures trade-offs between finite-state and minimal LLM-based approaches.

The most widespread method for hyphenation is pattern-based hyphenation, due to its high accuracy and generalization (Sojka and Sojka, 2023), efficiency (Sojka and Sojka, 2019a), small size, and potential applicability to any language with a hyphenated word list.

LLMs are considered overkill (computational waste) for simple rendering tasks on mobile devices/browsers, where a footprint of less than 50 KB is sufficient with pattern-based approaches.

Even though the best results for ever-developing languages and their new words are obtained by patterns generated by `patgen` from the large-scale word list of hyphenated words, a substantial portion of the 90 languages in the repository `hyph-utf8` rely on manually created rules. To develop hyphenation patterns, one needs to create a big, consistently hyphenated word list, together with heuristically optimized parameters for the `patgen` generation.

This paper describes a new flexible approach to the optimization of pattern generation that is based on Gaussian Process(*GP*) (Rasmussen and Williams, 2005). We start with the description of available word lists in Section 3. We proceed with the pattern generation process principles in Section 4. We describe the evaluation process and metrics used in Section 5. The design of our optimization by *GP* is explained, and the main results are described in Section 6. We conclude with the summary and future work in Section 7.

3 Datasets: Hyphenated Word Lists

Metelka and Sojka (2025) recently reported hyphenation benchmark with hyphenated word lists.

Table 1 shows the diversity of their collection: there are 18 word lists for 15 languages in several alphabets (Latin, Cyrillic, and Thai). There is significant variability between datasets (word lists) in

terms of their size, as well as average line lengths and density of hyphenation marks.

They stress that the quality of the data varies across datasets. Some of their datasets are curated and very consistent, such as *de/wortliste* (Lemberg, 2025) or *cssk/cshyphen* (Sojka and Sojka, 2021). On the other hand, there are no guarantees of quality in *wiktionary* datasets, as anyone can contribute to a Wiktionary entry, and these entries are typically not subject to professional review The dataset diversity will contribute to the evaluation of the new optimized pattern generation in the next section.

4 Pattern Generation with `patgen`

The `patgen` pattern generation tool (Liang, 1983) is part of the TEX Live 2025 distribution. It creates patterns from hyphenated word lists that are subsequently used in the TEX82 hyphenation algorithm. For efficient lookup, the algorithm uses a dynamic packed trie data structure for patterns.

A single run of the program consists of several phases. First, the *translate file* is read if present, and the characters (byte sequences) it contains are mapped to `patgen`'s internal codes. Then the input patterns are loaded. Subsequently, the algorithm iterates over the word list and decides which of the candidate patterns to include in the output collection. Finally, the selection is evaluated on the input word list to obtain values *good* (hyphenation point correctly identified), *bad* (hyphenation point identified on a non-hyphenated position of a word), and *missed* (hyphenation point not identified, but a word is hyphenated on that position). Speaking in statistical terms, these correspond to true positives, false positives, and false negatives, respectively.

The quality of patterns generated by `patgen` is influenced by the choice of algorithm parameters. We provide an overview in Table 2. It is possible to control the minimal distance of hyphenation points from the edge of the word (`\lefthyphenmin`, `\righthyphenmin`) and lengths of the patterns explored in the current run (*pat_start*, *pat_finish*). The power and the meaning of the generated patterns can be modified by specifying the hyphenation level (*hyph_start*, *hyph_finish*): patterns on higher levels overwrite those on lower levels during hyphenation; odd levels create hyphenating patterns, while even levels create inhibiting patterns (no hyphenation allowed). The parameters *good_wt*, *bad_wt*, and *thresh* play a key role in pattern selection. For a pattern to be accepted, the following inequality must hold (Liang and Breitenlohner, 2020):

$$\alpha * good\_wt - \beta * bad\_wt \geq thresh, \qquad (1)$$

where α denotes the number of positive examples (word split correctly by hyphenating pattern / previously wrong hyphenation fixed by inhibiting pattern), and β the number of negative examples (word split incorrectly by hyphenating pattern / previously correct hyphenation broken by inhibiting pattern) in the word list.

5 Pattern Generation Evaluation Metrics

We define the metrics for the evaluation of the hyphenation algorithm, pattern generation process, and ***parameter profiles*** that serve for advanced search techniques of parameter generation. Wherever possible, we opted for the same or similar metrics as used in prior work (Sojka and Sojka, 2023; Metelka and Sojka, 2025):

Precision P**:** the ratio of correct hyphenations out of all identified hyphenations, defined as $P = \frac{good}{good+bad}$. The values *good*, resp. *bad*, and *missed* are the amounts of correctly resp. incorrectly identified hyphenation points. This is the *primary metric*, as we do not want errors to appear in the automatically typeset documents.

Recall R**:** the ratio of correctly identified hyphenations out of all hyphenations in the data, defined as $R = \frac{good}{good+missed}$. This metric, also called **Coverage**, is the *secondary metric*, as allowing more hyphenation points may result in more uniform interword spacing and economic savings due to shorter documents.

F-score F**:** weighted harmonic mean of P and R (Rijsbergen, 1979). We prioritize P much more than R and use $F_{1/7}$-score, defined as $F_{1/7} = \frac{(1+1/7)^2 * P * R}{(1/7)^2 * P + R}$. The $F_{1/7}$-score thus serves as a joint measure of Precision and Recall, emphasizing the greater importance of not making errors over coverage.

Size: The number of trie nodes used to store the generated pattern is our *tertiary metric*. Trie depth limits the $O(1)$ time to find hyphenation points for a given word from patterns.

Table 1: Dataset collection overview: full size in kBytes for UTF-8 encoded files (Size), average word (segment) length in characters (#Lines)), average number of hyphenation points per word (#Hyphs), `\lefthyphenmin` and `\righthyphenmin` parameters (LRhyp), and the number of distinct lowercase characters (#Chars) occurring.

Language/Name	Size	#Lines	$\overline{\text{\#Lines}}$	$\overline{\text{\#Hyphs}}$	#Chars
cssk/cshyphen	10,646.4	835,145	11.10	2.54	50
cs/cshyphen_cstenten	8,202.6	597,170	11.83	2.79	48
cs/cshyphen_ujc	1,377.1	104,277	11.48	2.59	48
cs/wiktionary	632.0	53,962	9.95	2.16	48
de/wiktionary	12,821.6	807,813	15.01	2.96	86
de/wortliste	10,035.4	604,970	15.78	2.99	47
el/wiktionary	603.9	28,208	11.90	2.87	52
es/wiktionary	11,753.4	816,723	13.36	3.27	50
is/hyphenation-is	3,037.8	218,308	12.12	1.91	45
it/wiktionary	981.1	77,351	11.96	2.90	39
ms/wiktionary	13.6	1,247	8.52	1.81	69
nl/wiktionary	8,124.5	528,598	14.73	2.82	68
pl/wiktionary	1,713.6	141,916	10.75	2.20	52
pt/wiktionary	541.6	44,321	11.17	2.74	40
ru/wiktionary	3,795.6	152,608	13.10	2.74	51
th/orchid	3,567.1	34,868	40.69	6.02	95
tr/wiktionary	187.2	15,619	10.41	2.37	42
uk/wiktionary	351.6	15,701	11.79	2.65	41

Table 2: Overview of `patgen` parameters in pattern generation profiles

Name	Value range	Description
`\lefthyphenmin`	1 – 99	Length of word prefix without hyphenation, typically 1
`\righthyphenmin`	1 – 99	Length of word suffix without hyphenation, typically 1 or 2
hyph_start	1 – 9	Initial hyphenation level, typically 1
hyph_finish	1 – 9	Final hyphenation level, typically 4 or 5
pat_start, *pat_finish*	1 – 15	Minimal resp. maximal pattern length in given level, ranges typically increase in higher levels
good_wt	integer	Weight of examples in favor of the pattern
bad_wt	integer	Weight of examples against the pattern
thresh	integer	Evaluation threshold

Generalization: To assess the performance of the created patterns on unseen data, a 10-fold cross-validation is performed: We generate patterns from a train split of the data, measure their $F_{1/7}$ on the unseen test split, and average the values across the folds.

Parameter Profiles To compare different strategies of pattern generation, we define ***parameter profile*** as a set of adjustable parameter values that fully determine the `patgen` run, e.g., the quality of the resulting patterns. This means specifying pattern length window (*pat_start*, *pat_finish*) and weights (*good_wt*, *bad_wt*, *thresh*) for every hyphenation level as shown in Table 2.

Over the past three decades of experimentation with pattern generation, numerous profiles for various word lists have been developed. In Table 3, we show three profiles that gave the best results in prior work. One is adapted from earlier works (Sojka and Sojka, 2021, Table 2, *correct optimized* profile) *cshyphen*, and two from (Lemberg, 2025) *wortliste8*, and *wortliste* that contain the first 4 levels of the earlier.

Each profile defines a point p in `patgen` n-dimensional hyperparameter space, which sets the

Table 3: Parameter profile examples found by decades of empirical investigations. Parameters *pat_start* and *pat_finish* are combined into column "Lengths" and *good_wt*, *bad_wt*, and *thresh* into column "Params".

Profile	Level	Lengths	Params
cshyphen	1	1 3	1 5 1
	2	1 3	1 5 1
	3	2 6	1 3 1
	4	2 7	1 3 1
wortliste	1	1 3	2 3 1
	2	2 4	1 5 1
	3	3 5	1 6 1
	4	4 6	1 7 1

Profile	Level	Lengths	Params
wortliste8	1	1 3	2 3 1
	2	2 4	1 5 1
	3	3 5	1 6 1
	4	4 6	1 7 1
	5	5 12	1 8 1
	6	6 12	1 9 1
	7	7 12	1 9 1
	8	8 12	1 9 1

patterns' quality and performance. Hand tuning of profiles for each word list is very laborious and time-consuming work for `patgen` experts. A natural research question arises: can we find an *algorithmic way* better than trial and error to compute an optimal profile p, according to some objective function?

6 Optimal Pattern Generation with Gaussian Processes

The core idea of this paper is to define and use a Gaussian Process (Rasmussen and Williams, 2005; Snoek et al., 2012) to compute a profile p algorithmically in such a way that our three metrics are *jointly* optimized.

Formally, a Gaussian Process is specified by two functions: mean and covariance functions. We define our mean function m that represents the expected value of profile p as

$$m(p) = F_{1/7} - \text{trie\_penalty}\,. \tag{2}$$

We are looking for a profile p that maximizes $m(p)$ so that it is as close to 1 as possible. The rationale behind this design is as follows.

The first term, $F_{1/7}$, maximizes both primary and secondary metrics (precision and coverage). The preference for precision over recall is proper because false positives (incorrect hyphenations) are more severe than false negatives (missed hyphenations), which is typically the case for hyphenation problems. We followed the practice of prior work with this tradeoff setting, where bad hyphenation is 49 times worse than missed hyphenation.

The second term optimizes our tertiary metrics, pattern size, as it encourages smaller, more generalizable pattern sets. The trie node count correlates with the generalization. Smaller tries with shorter patterns and shorter contexts often generalize better to unseen words.

$$\text{trie\_penalty} = \text{trie\_weight}\frac{\text{trie\_nodes}}{\text{trie\_normalizer}} \tag{3}$$

The second critical component of a *GP* is the covariance function (kernel) that defines the relationship between input points p and p'. If the points are close to each other, their output values $m(p)$ and $m(p')$ should also be close. Our implementation of *GP* optimizer finds the best profile (and patterns) within n-dimensional parameter space. It uses `GaussianProcessRegressor` with `Matern` and `WhiteKernel` kernels from `sklearn.Gaussian_process`. It uses Upper Confidence Bound $UCB(x) = \mu(x) + \kappa \cdot \sigma(x)$ acquisition for exploration-exploitation balance.

Results We computed the best profiles using *GP* and compared them with the current State-of-the-Art (SOTA) profiles *cshyphen* and *wortliste*. We used the following optimization settings named *GPopt4(p)*:

objective: $F_{1/7}$ with trie size penalty as defined in Eq. (2).

number of generation levels: As there were no published examples of languages that would require a deeper hierarchy of exceptions than two inhibiting levels, we run *GP* for 4 levels. In each level, we have to decide parameters of Equation (1): *good_wt*, *bad_wt*, and *thresh*.

parameter ranges: To minimize the search space for 4 equations 1, we decided to force the same *thresh* for each level, bounded by $[1, 5]$. We search for *bad_wt* in $[1, 30]$, with fixed *good_wt* =3.

parameter dimension n: $n = 5$, *bad_wt* for four levels and one shared *thresh*.

pattern lengths ranges: We start with short patterns to continue with a longer context for inhibition. Range pairs (*min_length*, *max_length* for candidate patterns in each level are: $[(1,4), (2,5), (2,6), (2,7)]$.

trie_weight: 0.0005 as used in Eq. (3).

trie_normalizer: 25000 as used in Eq. (3); widespread natural language uses at least 25000 lemmas.

iterations: 100 at batch size 1.

UCB **kappa:** $\kappa = 2.5$ seems to be the right balance between exploration and exploitation.

To assess the generalization properties of the created patterns, a 10-fold cross-validation was performed. We generate patterns from a train split of the data, measure their performance on the unseen test split, and average the values across the folds.

We utilize the diverse dataset introduced by Metelka and Sojka (2025). In Table 4 we report the results of our experiments run on this dataset.

We have prepared several visualizations clarifying our *GP* method in Appendix A.

Key success The primary dataset *cssk/GPopt4(p)* for the paper (Czech+Slovak word list combined) shows that *GP* beats *all* baselines:

- *GPopt4(p)*: $F_{1/7} = 0.9992$, and trie size = 18888.
- Best baseline: $F_{1/7} = 0.9986$, and trie size = 25302 (*cshyphen* profile).
- Improvement: $+0.0006$ on $F_{1/7}$ (higher precision), and -6414 trie size (-25% smaller).

Trie-$F_{1/7}$ tradeoff Most *GP* solutions achieve higher training $F_{1/7}$ but larger tries that don't generalize as well.

Overfitting tendency The wide search space (*bad_wt* in [1,30]) leads to overfitting for some datasets. For production use, one might consider using tighter bounds (1–9) in matching profiles.

Summary statistics *GPopt4(p)* profile resulted in

$F_{1/7}$ **improved** for 11/18 datasets: *cssk/cshyphen*, *cs/cshyphen_cstenten*, *de/wiktionary*, *el/wiktionary*, *it/wiktionary*, *ms/wiktionary*, *nl/wiktionary*, *pl/wiktionary*, *pt/wiktionary*, *th/wiktionary*, *uk/wiktionary*

pattern tries smaller than best previous baseline for 7/18 datasets: *cssk/cshyphen*, *el/wiktionary*, *is/wiktionary*, *ms/wiktionary*, *nl/wiktionary*, *uk/wiktionary*, *es/wiktionary*

Efficiency and effectiveness remarks The entire optimization of Gaussian processes for all 18 word lists, along with follow-up cross-validation computations, could be rerun on a commodity laptop without a GPU in under a day.

While neural models such as Transformer/Bi-LSTM hyphenator may be pushed to achieve 99.9% accuracy, they require $1000 - 10000\times$ the memory.

Our method achieves a higher 99.999% accuracy with a 20 KB footprint, making it the only viable solution for embedding in mobile OS kernels or low-power e-readers. The inference speed is approximately $1000\times$ faster when measured by the number of hyphenated words per second.

Reproducibility All reported experiments are fully reproducible. Command

```
python -m scripts.optimize --lang cssk/cshyphen \
  --iterations 100 --batch-size 5 \
  --objective f17_trie --good-weight 3 \
  --max-bad-weight 30 --max-threshold 1 \
  --ucb-kappa 2.5 --trie-weight 0.0005 \
  --trie-normalizer 25000
```

does the *GP* optimization part on the Czecho-Slovak word list.

All reported experiments (software and data) are fully reproducible and available in the anonymous repository `https://anonymous.4open.science/r/hyph-bench-5F86/README.md`.

> "If I waited for perfection…I would never write a word."
> – Margaret Atwood

7 Conclusion and Future Work

We have designed a Gaussian Process optimized for hyphenated patterns generated by the `patgen` program.

We have demonstrated that automated *GP* optimization can find solutions that further improve `patgen` generation profiles hand-tuned for four

Table 4: Cross-validation results overview: $F_{1/7}$-score (rounded to 4 decimal points) and the number of nodes in the pattern-storing trie (rounded to the lowest higher integer) for *cshyphen*, *wortliste*, and *GPopt4(p)* parameter profiles, averaged over the 10 cross-validated folds. The 5 parameters optimized by *GPopt4(p)* are 4 *bad_wt* for each level and *thresh*.

Language/Name	*cshyphen*		*wortliste*		*GPopt4(p)*						
	$F_{1/7}$	trie size	$F_{1/7}$	trie size	$F_{1/7}$	trie size	5 params				
cssk/cshyphen	0.9986	25302	0.9981	21685	0.9992	18888	2	2	6	12	1
cs/cshyphen_cstenten	0.9444	12052	0.9243	12121	0.9586	11929	3	1	6	7	1
cs/cshyphen_ujc	0.9830	17046	0.9803	14736	0.9820	26653	23	1	30	16	1
cs/wiktionary	0.9759	10643	0.9744	8330	0.9742	17574	27	2	29	25	1
de/wiktionary	0.9983	32394	0.9977	29449	0.9985	45653	20	2	22	1	1
de/wortliste	0.9775	47544	0.8985	43214	0.9452	47397	2	3	5	1	1
el/wiktionary	0.9875	3602	0.9866	3223	0.9880	3524	4	2	30	1	1
es/wiktionary	0.9998	2023	0.9998	1784	0.9998	1902	1	1	23	1	1
is/hyphenation-is	0.9839	32284	0.9802	28664	0.9825	27324	1	5	1	1	1
it/wiktionary	0.9970	3593	0.9968	2875	0.9971	4158	30	1	30	21	1
ms/wiktionary	0.9296	796	0.9244	721	0.9302	937	15	1	1	11	1
nl/wiktionary	0.9972	30859	0.9963	27514	0.9972	31798	3	2	10	1	1
pl/wiktionary	0.9913	8564	0.9904	7262	0.9915	15062	30	1	24	1	1
pt/wiktionary	0.9969	1761	0.9970	1422	0.9971	1925	13	1	23	1	1
ru/wiktionary	0.9270	53820	0.9212	42164	0.9252	101559	30	8	29	1	1
th/orchid	0.9622	28795	0.9488	24258	0.9681	38955	30	6	30	1	1
tr/wiktionary	0.9920	1406	0.9916	1273	0.9915	2064	30	1	24	1	1
uk/wiktionary	0.9602	4741	0.9564	4272	0.9615	4514	3	1	3	11	1

decades. With consistently marked and curated hyphenated word lists, the resulting solution might be very, very close to the optimal one.

These results set the floor for improvements and savings in all typesetting systems and rendering engines that hyphenate language texts.

We hope that the new method will stimulate the research and development of new word lists or further pattern generation strategies. The reported results for *cssk/GPopt4(p)* are encouraging for the development of patterns for sets of languages that share phonetic rules of syllabification and hyphenation, e.g., for universal Slavic hyphenation.

In future work, to achieve the above vision, the next steps are:

further *GP* profile development Pattern generation could still be further improved (speed) by *annealing* techniques (Kirkpatrick et al., 1983; Ingber, 1989).

word list development Word lists could be extended from available sources with the techniques of bootstrapping (Sojka and Sojka, 2021, 2019b) and transfer learning from (Sojka, 2025).

pattern generation for big alphabets Universal hyphenation and segmentation preparation (Shao et al., 2018) will need a new version of the `patgen` program capable of working with Unicode-size alphabets.

Limitations

The current optimization parameters of the Gaussian process need further investigation to minimize the heuristic setting of parameter ranges. One possibility is to use wider parameter ranges with some of the efficient annealing techniques, e.g., very fast simulated re-annealing by Ingber (1989). We are aware that the current set of hyphenated word lists is limited, as is the size and consistency of data from Wiktionary. For universal use in any natural language, the *GP* should be tested on a bigger database of hyphenated words.

While we utilized AI assistance (Claude 4.5, Gemini 3 Pro) for copy-editing and code checking in accordance with the ACL 2026 Policy on Publication Ethics, all scientific claims and methodological designs were verified by the human authors.

Table 5: Comparison of profiles: *GP* optimized *GPopt4(p)* versus previous best base SOTA profiles *cshyphen* or *wortliste*

Language/Name	*GP* $F_{1/7}$	*GP* trie	Best base $F_{1/7}$	Best base profile	Base trie	$F_{1/7}$ better?	
cssk/cshyphen	**0.9992**	**18888**	0.9986	cshyphen	25302	**YES**	(+0.0006)
cs/cshyphen_cstenten	**0.9586**	**15278**	0.9444	cshyphen	12052	**YES**	(+0.0142)
cs/cshyphen_ujc	0.9820	26653	0.9830	cshyphen	17046	NO	(−0.0010)
cs/wiktionary	0.9742	17574	0.9759	cshyphen	10643	NO	(−0.0017)
de/wiktionary	**0.9985**	45653	0.9983	cshyphen	32394	**YES**	(+0.0002)
de/wortliste	0.9452	47397	0.9775	cshyphen	47544	NO	(−0.0323)
el/wiktionary	**0.9880**	**3524**	0.9875	cshyphen	3602	**YES**	(+0.0005)
es/wiktionary	0.9998	1902	0.9998	cshyphen	2023	NO	(−0.0000)
is/hyphenation-is	0.9825	27324	0.9839	cshyphen	32284	NO	(−0.0014)
it/wiktionary	**0.9971**	4158	0.9970	cshyphen	3593	**YES**	(+0.0001)
ms/wiktionary	**0.9302**	**711**	0.9296	cshyphen	796	**YES**	(+0.0006)
nl/wiktionary	**0.9972**	**30097**	0.9972	cshyphen	30859	**YES**	(+0.0000)
pl/wiktionary	**0.9915**	15062	0.9913	cshyphen	8564	**YES**	(+0.0002)
pt/wiktionary	**0.9971**	1925	0.9970	wortliste	1422	**YES**	(+0.0001)
ru/wiktionary	0.9252	101559	0.9270	cshyphen	53820	NO	(−0.0018)
th/orchid	**0.9681**	38955	0.9622	cshyphen	28795	**YES**	(+0.0059)
tr/wiktionary	0.9915	2064	0.9920	cshyphen	1406	NO	(−0.0005)
uk/wiktionary	**0.9615**	**4514**	0.9602	cshyphen	4741	**YES**	(+0.0013)

References

Gosse Bouma. 2003. Finite State Methods for Hyphenation. *Natural Language Engineering*, 9(1):5–20.

Jan Franěk. 2025. Transformer-Based Word Hyphenation. Master's thesis, Masaryk University, Brno, Faculty of Informatics.

Lester Ingber. 1989. Very fast simulated re-annealing. *Mathematical and Computer Modelling*, 12(8):967–973.

Scott Kirkpatrick, C. Daniel Gelatt Jr, and Mario P. Vecchi. 1983. Optimization by simulated annealing. *Science*, 220(4598):671–680.

Donald E. Knuth. 1986. *The TEXbook*. Addison-Wesley, Reading, Massachusetts. See Appendix H for the hyphenation algorithm.

Jacob Krantz, Maxwell Dulin, and Paul De Palma. 2019. Language-Agnostic Syllabification with Neural Sequence Labeling. In *18th IEEE International Conference On Machine Learning And Applications, ICMLA 2019, Boca Raton, FL, USA, December 16–19, 2019*, pages 804–810. IEEE.

Werner Lemberg. 2025. A database of German words with hyphenation information. Wordlist version with last change of Sep 20th, 2025.

Franklin M. Liang. 1983. *Word Hy-phen-a-tion by Com-put-er*. Ph.D. thesis, Dept. of Computer Science, Stanford University.

Franklin M. Liang and Peter Breitenlohner. 2020. PATtern GENeration program for the TEX82 hyphenator.

Ondřej Metelka and Petr Sojka. 2025. Hyph-bench: Benchmark Dataset of Hyphenated Words for Generating Hyphenation Patterns. In *Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2025*, pages 173–185, Brno, Czech Republic. Tribun EU.

Carl Edward Rasmussen and Christopher K. I. Williams. 2005. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Cornelis J. Van Rijsbergen. 1979. *Information Retrieval*. Butterworths.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the ACL (Volume 1: Long Papers)*, pages 1715–1725. ACL.

Yan Shao, Christian Hardmeier, and Joakim Nivre. 2018. Universal Word Segmentation: Implementation and Interpretation. *Transactions of the Association for Computational Linguistics*, 6:421–435.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Proceedings of the 25th International Conference on Neural Information Pro-*

cessing Systems (NIPS'12), volume 25, pages 2951–2959. Curran Associates Inc.

Ondřej Sojka. 2025. Transfer Learning of Slavic Syllabification for Hyphenation Patterns. Bachelor Thesis, Masaryk University, Brno, Faculty of Informatics (advisor: Pavel Šmerk), https://is.muni.cz/th/s17d6/?lang=en.

Ondřej Sojka and Petr Sojka. 2023. Towards Perfection of Machine Learning of Competing Patterns: The Use Case of Czechoslovak Patterns Development. In *Proceedings of the 17th Workshop on Recent Advances in Slavonic Natural Language Processing, RASLAN 2023, Kouty nad Desnou, Czech Republic, December 8–10, 2023*, pages 113–120, Brno. Tribun EU.

Petr Sojka. 1995. Notes on Compound Word Hyphenation in TEX. *TUGboat*, 16(3):290–297.

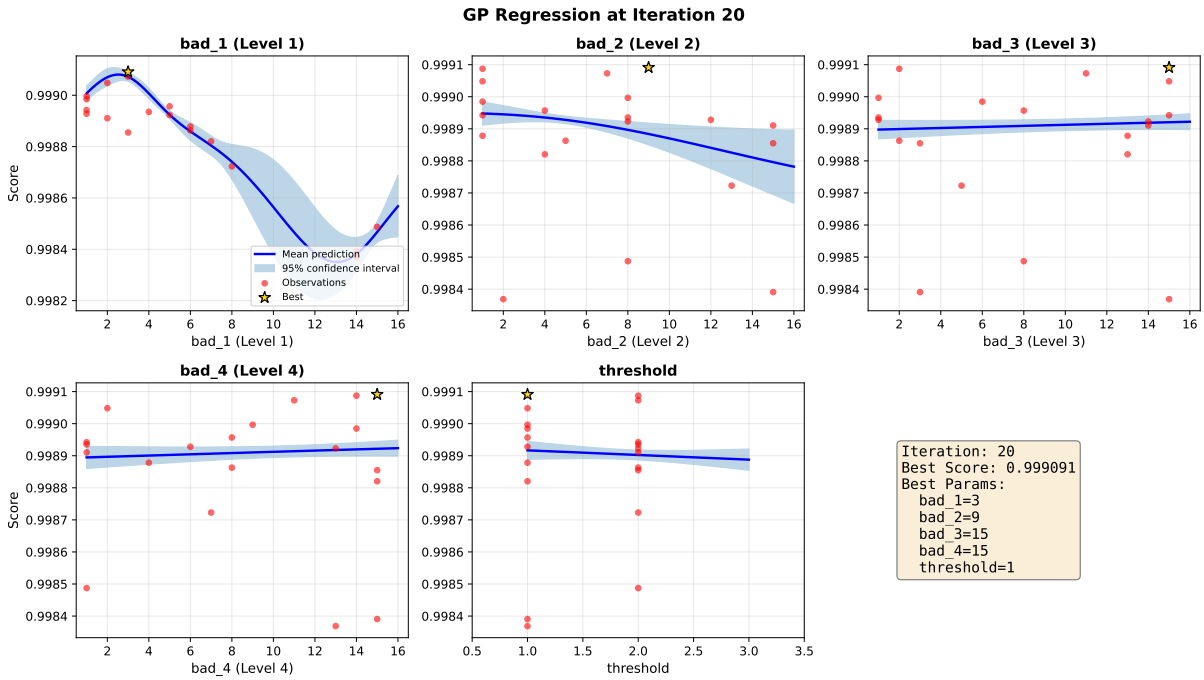
Petr Sojka and Ondřej Sojka. 2019a. The Unreasonable Effectiveness of Pattern Generation. *TUGboat*, 40(2):187–193.

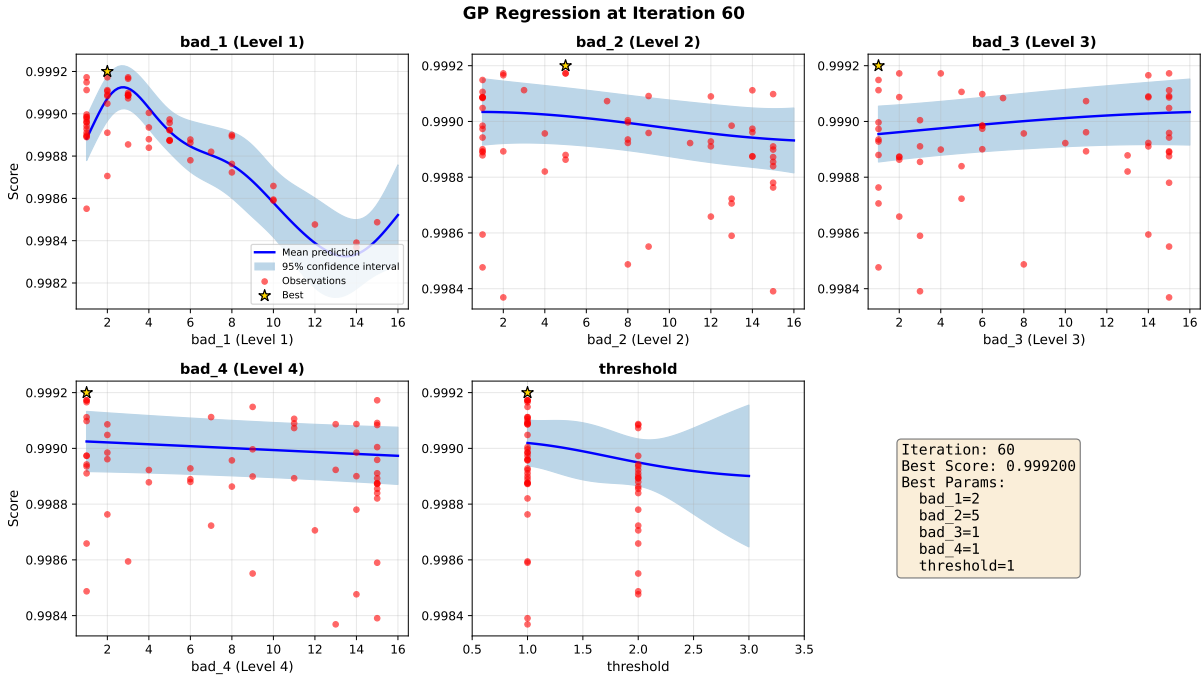
Petr Sojka and Ondřej Sojka. 2019b. Towards Universal Hyphenation Patterns. In *Proceedings of Recent Advances in Slavonic Natural Language Processing—RASLAN 2019*, pages 63–68, Karlova Studánka, Czech Republic. Tribun EU. https://is.muni.cz/publication/1585259/?lang=en.

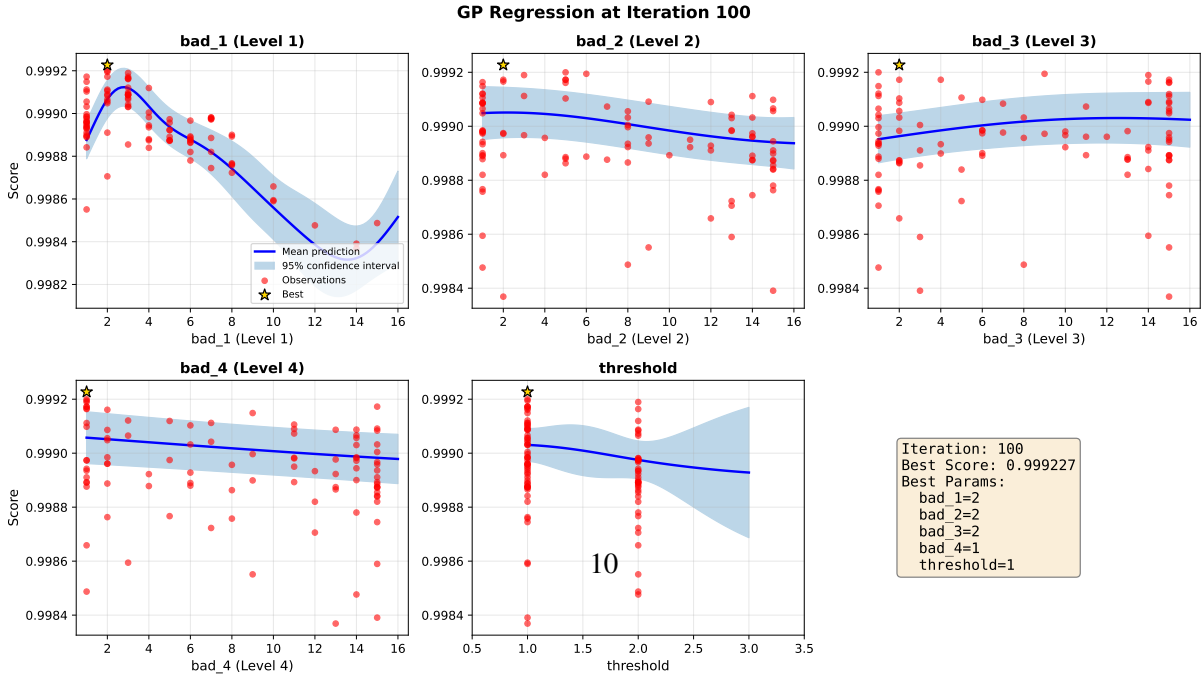
Petr Sojka and Ondřej Sojka. 2021. New Czechoslovak Hyphenation Patterns, Word Lists, and Workflow. *TUGboat*, 42(2).

A Visualizations of Gaussian Processes

To better understand the parameters and results of Gaussian Processes, we provide visualizations in figures 2, 3, 4, and 5.


GP Regression at Iteration 20
bad_1 (Level 1)
bad_2 (Level 2)
bad_3 (Level 3)
bad_4 (Level 4)
threshold
Score
Mean prediction
95% confidence interval
Observations
Best
Iteration: 20
Best Score: 0.999091
Best Params:
bad_1=3
bad_2=9
bad_3=15
bad_4=15
threshold=1


GP Regression at Iteration 60
bad_1 (Level 1)
bad_2 (Level 2)
bad_3 (Level 3)
bad_4 (Level 4)
threshold
Score
Mean prediction
95% confidence interval
Observations
Best
Iteration: 60
Best Score: 0.999200
Best Params:
bad_1=2
bad_2=5
bad_3=1
bad_4=1
threshold=1


GP Regression at Iteration 100
bad_1 (Level 1)
bad_2 (Level 2)
bad_3 (Level 3)
bad_4 (Level 4)
threshold
Score
Mean prediction
95% confidence interval
Observations
Best
Iteration: 100
Best Score: 0.999227
Best Params:
bad_1=2
bad_2=2
bad_3=2
bad_4=1
threshold=1

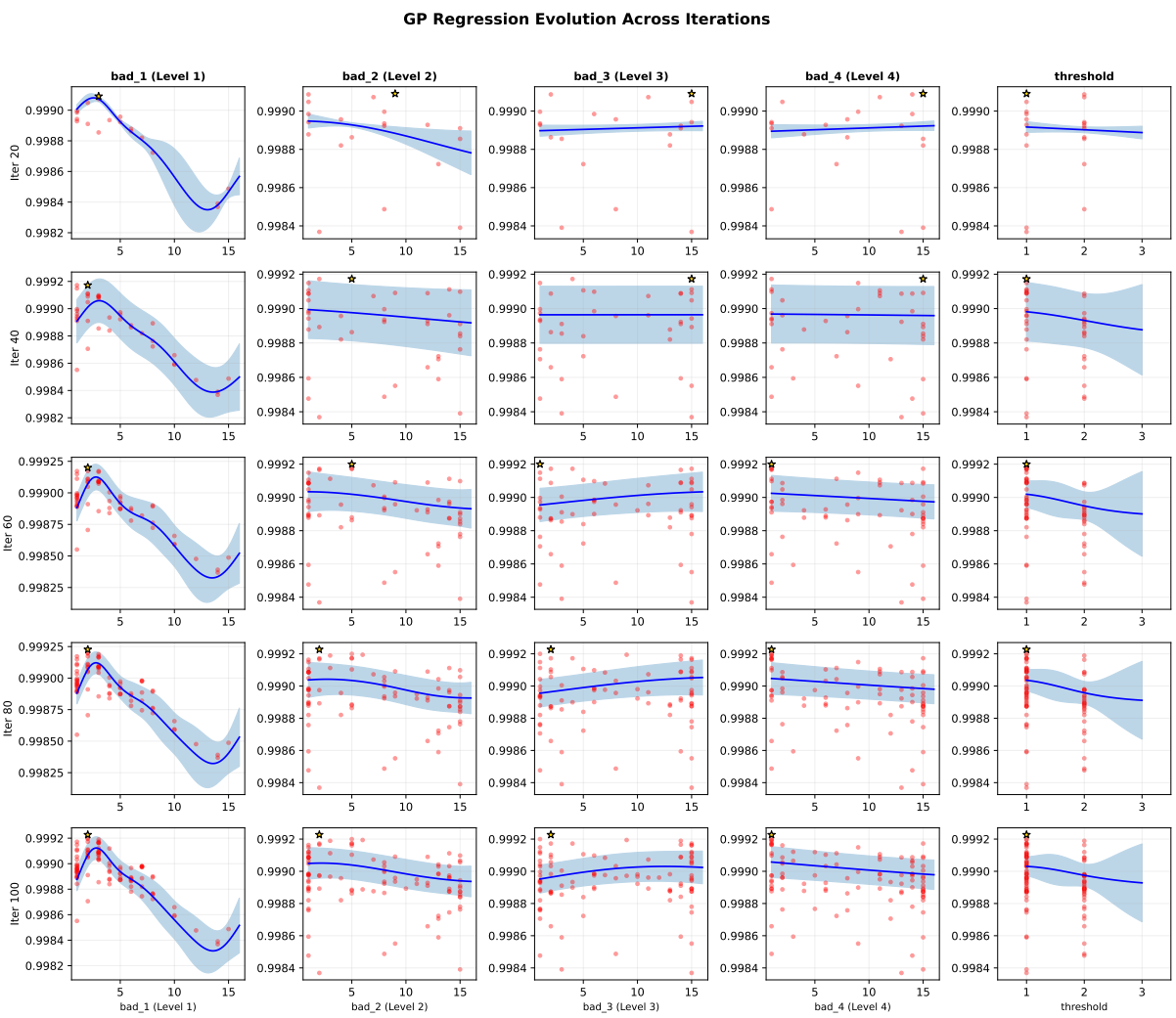


Figure 3: *GP* iterations 20, 60 and 100 for *cssk* dataset.

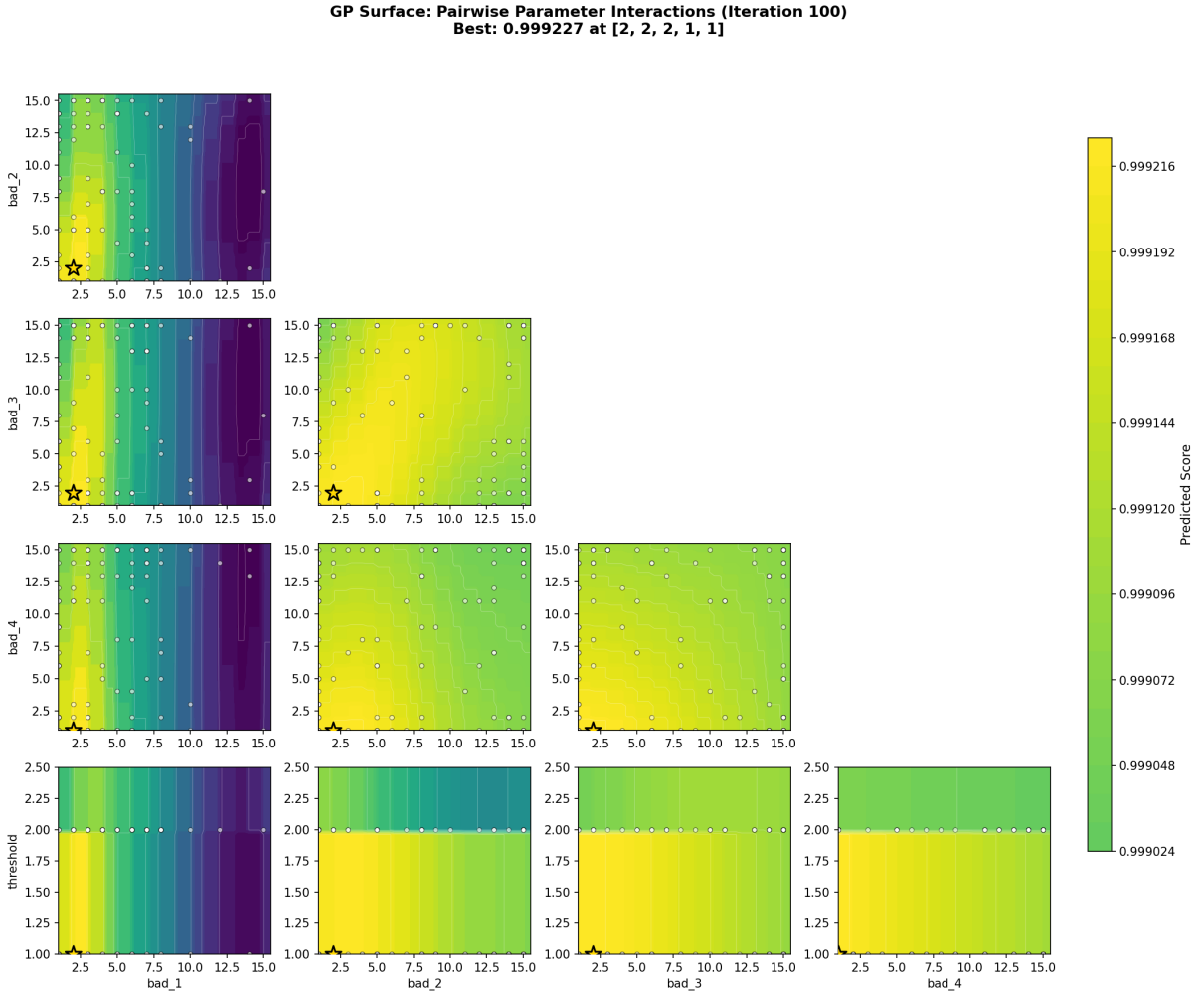


Figure 4: *GP* pairwise parameter interactions at iteration 100 for *cssk* dataset. "Stars" represent the optimum found.

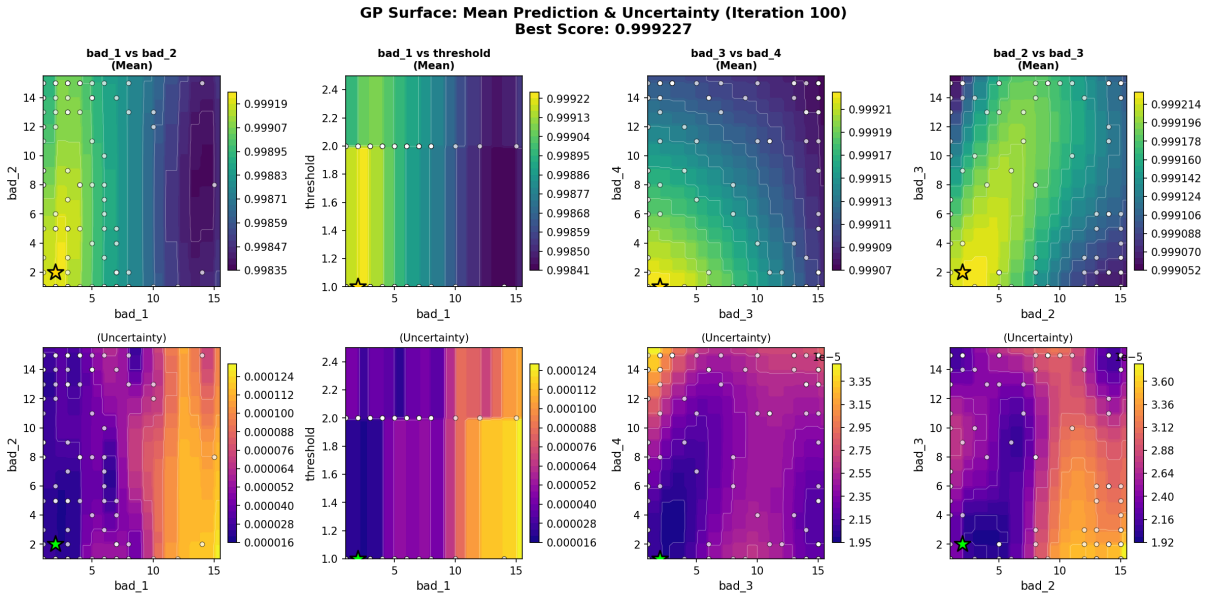


Figure 5: *GP* mean prediction and uncertainty at iteration 100 for *cssk* dataset.