Performance Testing of Stroke Gesture Recognizers with Gester

NATHAN MAGROFUOCO and JEAN VANDERDONCKT, Université catholique de Louvain, Belgium
PAOLO ROSELLI, Università degli Studi di Roma "Tor Vergata", Italy

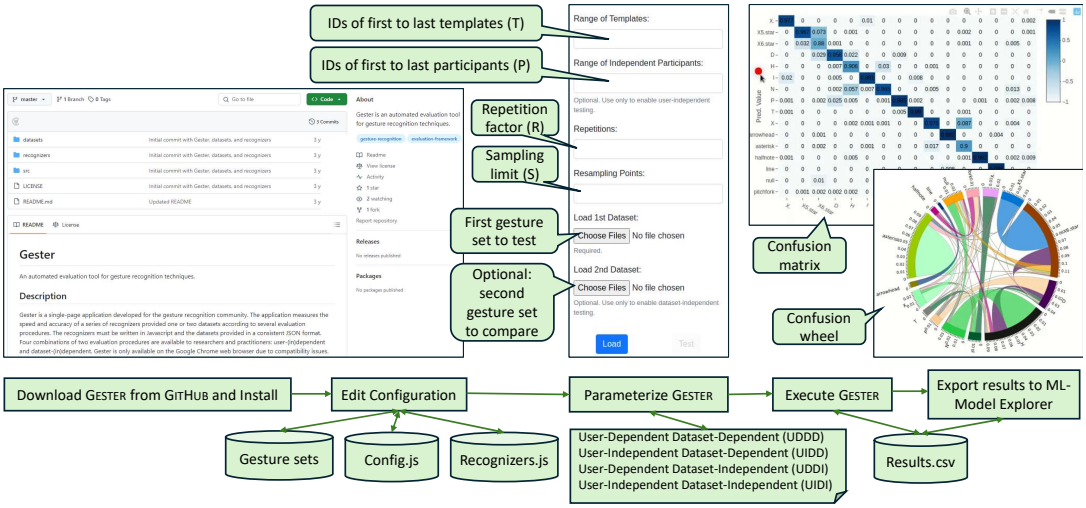


Fig. 1. Automated evaluation of recognizers with Gester: downloading from GitHub and installing, editing the configuration (*e.g.,* which recognizers), specifying the testing parameters (*e.g.,* one gesture set for internal evaluation or two gesture sets for comparison), executing Gester, and exporting results for information visualization.

The performance of state-of-the-art stroke gesture recognizers has been extensively tested against accuracy and speed in the user-dependent or user-independent procedure on a wide variety of gesture sets. These tests were carried out under heterogeneous experimental conditions, making the results non-comparable from one source to another, and calling for a consistent assessment that ensures reproducibility of the results. When a new gesture set should be incorporated in a gesture-based user interface and/or when a new recognizer is made available, this gesture set needs to be tested with existing and new recognizers to determine which recognizer offers the best performance in a given context of use. To this end, this technical note presents Gester, a software that automates the performance testing of stroke gesture recognizers in terms of accuracy (through three recognition rates) and speed (through four execution times). It exports data for visualizing confusion matrices and confusion wheels. We elaborate on typical use cases supported by Gester, which include the two traditional testing procedures, *i.e.,* user-dependent and user-independent scenarios, and provide two new ones, *i.e.,* dataset-dependent and dataset-independent, to test the performance of recognizers under new conditions. Then, we illustrate a series of activities.

Authors' Contact Information: Nathan Magrofuoco, nathan.magrofuoco@uclouvain.be; Jean Vanderdonckt, jean.vanderdonckt@uclouvain.be, Université catholique de Louvain, Louvain Research Institute in Management and Organizations (LouRIM), Louvain-la-Neuve, Belgium; Paolo Roselli, roselli@mat.uniroma2.it, Università degli Studi di Roma "Tor Vergata", Roma, Italy.

ACM 2573-0142/2025/6-ARTEICS009
https://doi.org/10.1145/3734186

CCS Concepts: • **Software and its engineering** → *Software verification*; *Acceptance testing*; Software testing and debugging; • **Human-centered computing** → **Graphical user interfaces**; Touch screens; **Gestural input**; User interface programming.

Additional Key Words and Phrases: Benchmarking, Gesture datasets, Gestural interaction, Gesture recognition, Gesture user interfaces, Performance testing

ACM Reference Format:
Nathan Magrofuoco, Jean Vanderdonckt, and Paolo Roselli. 2025. Performance Testing of Stroke Gesture Recognizers with Gester. *Proc. ACM Hum.-Comput. Interact.* 9, 4, Article EICS009 (June 2025), 25 pages. https://doi.org/10.1145/3734186

1 Introduction

In software engineering, *performance testing* [43] is a non-functional software testing technique that determines how the stability, speed, scalability, and responsiveness of an application subject to a given workload. It entails running the software through its paces using manual or automated tools [23] to evaluate one or more properties of interest, typically software quality factors such as those defined in the ISO 25010 Square standard [52], including efficiency and usability [53].

More specifically for gesture-based user interfaces (UIs) computing [15], the accuracy [25], its end-user acceptance [38], and the speed of gesture recognizers [42, 61] are two quality factors that are typically evaluated in user-dependent and user-independent procedures [39] on a wide variety of gesture sets [79]. Given any new gesture set, researchers and practitioners must test the performance of one or more recognizers to identify the best-fitting algorithm for a particular gesture-based UI in a given context of use [21]. To our knowledge, there is yet no such tool that supports or automates testing the performance of one or many recognizers on one or multiple datasets as a main goal. This kind of performance testing can be found in a large number of papers (*e.g.,* LVS [16], $1 [87], $N [3], Protractor [44], $N-Protractor [4], 1F [77], $P [80], $P+ [78], $Q [82], Match & Conquer [56], PennyPincher [71], Jackknife [72], !FTL [75], G-Gene [13] are representative examples (see [46, 63] for a general survey and a survey specific for children, respectively) that introduce a gesture recognizer to position its performance against other state-of-the-art candidates [72].

To this end, we present Gester, a software that automates the performance testing of two or many stroke gesture recognizers on one or two datasets at a time (Figure 1) in terms of accuracy (through three recognition rates) and speed (through four execution times). It exports data for visualizing confusion matrices and confusion wheels. Gester supports the two traditional procedures, *i.e.,* user-dependent and user-independent [80], combined with two additional ones *i.e.,* dataset-dependent and dataset-independent, to evaluate recognizers on new conditions.

Moreover, the performance testing of recognizers, as reported in papers, is mostly performed in an opportunistic way that is very challenging, if not impossible, to reproduce without reusing the authors' software and hardware configurations. Even when the software is publicly available, the experimental conditions, such as software/hardware conditions, are not always consistent, which may lead to different results, at least in terms of execution times. Gester stores testing parameters (Fig. 1) in a configuration file that can be reused and replayed to support the three ACM badges[1]: *repeatability*, *reproducibility*, and *replicability*, where someone else wishes to re-run the same performance testing on another computer, preferably the one of the end-user, perhaps with different computational powers, or in other experimental conditions specified in RepliGES [27].

The remainder of this technical note is organized as follows: section 2 positions Gester with respect to other categories of software; section 3 describes the implementation of Gester, including its software architecture, how a dataset is imported and how results are exported; section 4 expresses the procedures for performance testing in user-dependent and user-independent scenarios to introduce two new ones, allowing four overall testing procedures (Figure 1). Based on these procedures, section 5 illustrates typical activities of Gester with concrete examples, section 6 concludes this technical note and suggests some future avenues for this work.

[1]See https://www.acm.org/publications/policies/artifact-review-and-badging-current

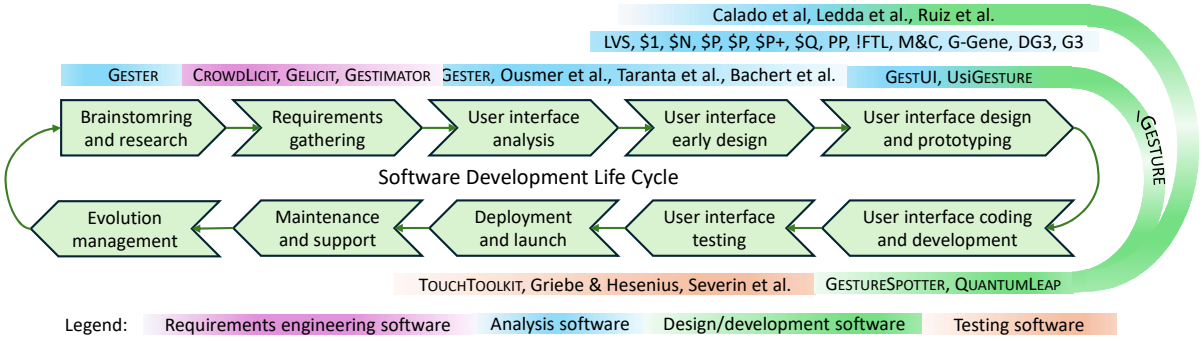


Fig. 2. Four categories of software supporting stages on the development lifecycle.

2 Related Work

Testing the performance of a stroke gesture recognizer is one stage in the user interface (UI) development lifecycle [68] in which four categories of support software can be distinguished for covering the various stages [94] (Fig. 2):

(1) *Software addressing the Requirements Engineering and beyond*: in software engineering, *requirements engineering* involves defining, documenting, and maintaining both functional and non-functional requirements to meet end users' needs. To contribute to this category, CROWDLICIT [1] and GELICIT [49] are expected to collect user-defined gestures that correspond to the end user's preferences and needs, being distributed in time and space. GELICIT [49] is a cloud computing platform that supports gesture elicitation studies [84**?**] distributed in time and space. Xia et al. [89] defined 13 factors for addressing gesture requirements in four categories: situational (*e.g.,* context), cognitive (*e.g.,* discoverability), physical (*e.g.,* ergonomics), and system (*e.g.,* accuracy). GESTIMATOR [93] measures the similarity between a gesture and a set of pre-recorded templates to determine the appropriate gesture classes that are different enough from each other.

(2) *Software addressing the Analysis*: in software engineering, *analysis* should devise a logical model of the proposed application with its UI. One of these goals consists of analyzing and determining the most appropriate gesture set, preferably of user-defined gestures, that can be efficiently recognized, which is the main purpose of GESTER. This goal can also be investigated for research and development purposes in the Brainstorming and research stage, when GESTER is more intended to be used by researchers and experimenters than practitioners. When available gestures are not numerous enough to feed a recognizer, Taranta et al. [70] and [6] design and evaluate synthetically-generated gestures to ensure that they are "human-like" [41]. These actions span until design.

(3) *Software addressing the Design and Development*: in software engineering, *design* consists of defining the overall software architecture, its software components, data flows, and user interactions. All recognizers are candidates to be adopted during this design stage, ranging from the oldest ones such as Rubine [58] to the most recent ones such as μC_1, μc_2 [12], RNN for real-time recognition [40]. When these recognizers are already integrated into a platform [59], a framework [66], a toolkit [28], a programming language [32], or an integrated development environment [65], the design can be pursued to rapid prototyping of a gesture-based UI (as initiated in [3]) and the preliminary steps of development. A representative example is IGESTURE[2] [65], a Java-based gesture recognition framework that focuses on extensibility and cross-application reusability. This integrated solution includes tools for covering the major stages of gesture recognition, ranging from creating and managing gesture sets for testing to optimizing new or existing

[2]See http://www.igesture.org/

recognizers. A data structure specifically tailored to represent gestures enables the representation of single gestures and groups of gestures and allows the gestures to be processed by a wide range of recognizers. It can span until testing as well as soon as the iGesture Test Bench[3] enables the manual configuration of testing a recognizer for one gesture set and one test set at run-time. UsiGesture [8] is a structured method for engineering stroke gestures in graphical UIs with some guidance to engineers, programmers, and designers. It structures a gesture set and associates it with a recognizer, but does not incorporate any tool automation for performance testing like Gester offers. GestUI [55] is similar for information systems.

(4) *Software addressing the Testing*: in software analysis, *testing* uses automated test scripts to perform interactions with the UI so that the tester only executes the script and interprets the results [23]. For example, Squish[4] automatically tests common gestures such as touch, flick, swipe, and pinching. Yet, this stage is more complex for other gestures than the common ones and is more sophisticated for gesture-based UI than for graphical UIs [31]. Hesenius et al. [31] extended the test automation framework Calabash to translate formally specified touch gestures into corresponding human-like touch events injected in test case execution. Griebe et al. [30] presented a life cycle that integrates gesture input information into user acceptance tests and uses a sensor simulation engine to automate test case execution. GestureSpotter [64] consists of a gesture-spotting architecture that encapsulates a software architecture into a GUI and a series of API calls for quickly building and evaluating real-time gesture-spotting recognizers for augmented or virtual reality. Yet, out-of-distribution situations, such as situations where real-time gestures are different from design-time gestures), require a more profound approach to preserve the ability to test gesture-based UIs. TouchToolkit [37] consists of a toolkit that provides a touch interaction record and playback system that simplifies testing and debugging as well as automating unit tests to validate multi-touch gesture-based interactions. Its device-independent API enables introducing of new device support and a new recognizer. Severin et al. [62] test the robustness of state-of-the-art machine learning approaches in Out-of-Distribution situations, which are cases where input data are very different from the data the classifier was trained on.

Tu et al. [74] empirically compared sets of stroke gestures produced with varying degrees of complexity and in different target sizes using both the finger and the pen. Closer to our area, Erazo and Pérez Medina [22] compared five recognizers, *i.e.,* $1 [87], $P [80], $Q [82], !FTL [75], and Penny Pincher [70]), on three gesture sets, *i.e.,* NicIcon [86], HHReco [33], and the Utopian Alphabet[5] in a user-independent scenario. This testing was performed manually with specifically developed ad hoc software, making it difficult to reuse for other testing procedures and conditions. Similarly to this study, Filho et al. [24] compared the effectiveness of various machine learning algorithms for real-time gesture recognition to find the optimal way to identify static hand gestures, as well as the optimal sample size [76] for training the recognizer. These initiatives are laudable in that they shed light on the quality of recognizers in the best conditions, but the resulting code is very difficult to reuse outside the context in which it was conceived. Moreover, they do not allow us to follow a precise procedure [33, 54, 76].

In sum, the aim of Gester is to test the performance of gesture recognizers in a parameterizable way (*e.g.,* by choosing recognizers, datasets, by assigning parameters) rather than fixed (*e.g.,* some studies freeze experimental conditions without it being possible to change them), reproducible rather than opportunistic (*e.g.,* it is not possible to reproduce the test under the same conditions), but above all systematic rather than predefined and homogeneous (*e.g.,* using procedures consistently and visibly) rather than heterogeneous. This can be carried out during the analysis and design stages and during the brainstorming stage for research, preliminary investigation, and free exploration. Gester therefore does not cover the subsequent stages, such as unit test automation [30, 31, 37, 85].

[3]See http://www.igesture.org/tutorial_testbench.html
[4]See https://www.qt.io/product/quality-assurance/squish.
[5]See https://www.omniglot.com/conscripts/utopian.htm.

3 Implementation

Since all recognizers from the $-family [3, 82], !FTL [75], and Jackknife [72] have been publicly released in Javascript[6], a language suited for the fast prototyping of gesture-based UIs of interactive applications, we implement Gester as a single-page application meant to run locally on the evaluator's browser. Therefore, Gester is built in HTML5[7], Cascading Style Sheets Level 3 (CSS3)[8], and Javascript. The application requires local access to a series of recognizers, datasets, and a configuration file. The recognizers must be implemented in Javascript, whereas the datasets must be defined in a consistent JSON format. The configuration file is merely a Javascript file that must be customized by the evaluator to map the relevant evaluation procedures to the corresponding recognizer functions. Gester reports results in terms of accuracy, speed, and confusion matrices for each recognizer in each selected procedure, given one or two datasets and a series of parameters (the number of templates T, the number of participants P, and the number of repetitions R) entered by the evaluator via the single-page application.

3.1 Software Architecture

Gester consists of an HTML file, a CSS file, and a series of Javascript files to define the evaluation procedures and the recognizers supported by the application. Figure 3 illustrates how files interact with each other:

- **Index.** The index file displays the application interface in which the evaluator sets up a series of parameters in addition to the configuration file (Figure 1): how many templates (T) and independent participants (P) must be used to train the recognizers, how many repetitions (R) are required for each testing participant, and to how many points gestures must be resampled (N) [76]. Gester expects a single value for the parameter R and N, whereas it expects a range of values for the parameters T and P. The application supports two operators to define a range of values. The operator "+" is used to enter two distinct numbers (*e.g.,* "1+4+8"

[6]See https://developer.mozilla.org/en-US/docs/Web/JavaScript.
[7]See https://www.w3schools.com/html/.
[8]See https://www.tutorialrepublic.com/css-tutorial/.

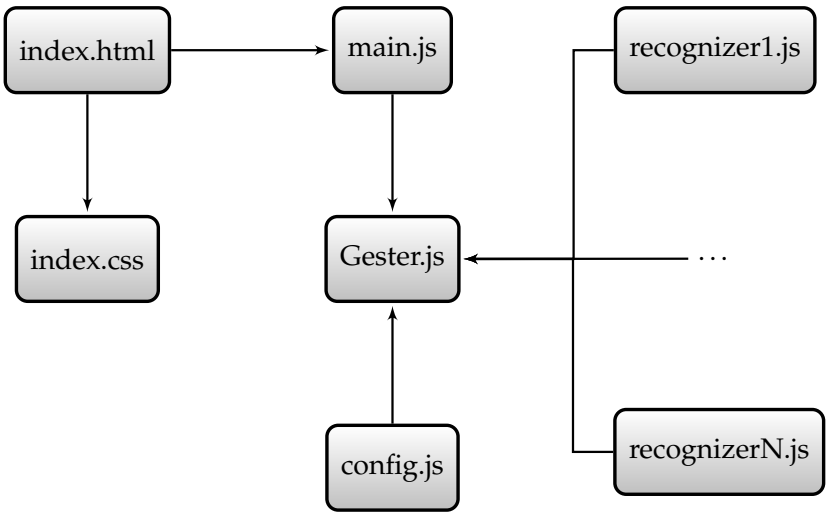


Fig. 3. Software architecture of Gester.

means that the parameter will be tested in three conditions: 1, 4, and 8). The operator "-" is used to enter all numbers within the range (*e.g.,* "1-3" means that the parameter will be tested with three conditions: 1, 2, and 3). Both operators can be combined into a single equation to enter multiple values (*e.g.,* "1-4+8" means that the parameter will be tested with five conditions: 1, 2, 3, 4, and 8). The index file is also used to import the relevant datasets and provides two buttons, *i.e.,* load and test, to load the dataset(s) in memory and start the evaluation procedures, respectively. The "test" button is deactivated until at least one dataset is loaded in memory. We discuss the import of the datasets in the next subsection.

- **Config.** The configuration file must be customized before loading the index file in the web browser. It consists of a list of the available recognizers and, for each recognizer, it is responsible for mapping its functions to the corresponding evaluation steps. There are six evaluation steps concerned by this mapping:

(1) *setupPoint* describes how to instantiate a gesture point for all recognizers (*e.g.,* since $1 was built for uni-strokes, its Point constructor does not include a *strokeId* parameter, whereas μV is a multi-stroke recognizer that requires a *strokeId* for all points).
(2) *setupSample* defines how to instantiate a gesture sample for all recognizers (*e.g.,* $1 stores each sample as an array of points, whereas $N uses an array of arrays with one array per stroke).
(3) *setupRecognizer* describes how to instantiate a recognizer and provides it with the right parameters (*e.g.,* its resampling rate).
(4) *addTemplate* defines how to add a training sample, or template, into each recognizer. This method returns the time required to add the template for each recognizer.
(5) *trainRecognizers* must be used by the recognizers that need to be trained explicitly (*e.g.,* Rubine and Jackknife await for an explicit call before starting their training, whereas the $-family pre-processes each template as soon as they are sent into the algorithm by the *addTemplate* method). This method returns the time required for training each recognizer.
(6) *recognize* starts the classification of a candidate and returns the best matching template along with two-time measurements, one for pre-processing the candidate and the other for classification against all available templates.

- **Gester.** Gester is the core Javascript file responsible for running the selected evaluation procedures, and gathering, averaging, and reporting all results into separate CSV files that are easy to import into most statistical programs. The application prints one file per type of result.
- **Main.** As soon as the evaluator has pressed the "Test" button in the application interface, this small file instantiates a Gester object and starts the evaluation of all recognizers on the provided datasets. It is also responsible for loading the datasets in memory via the "Load" button.

3.2 Import Datasets

The evaluation process starts with a dataset, *i.e.,* a set of gesture samples performed by one or multiple participants. Gester requires a specific structure for the files that make up a data set and a specific format for all samples.

- **Structure.** A dataset is stored as a folder that contains all gesture samples. This folder can be divided into subfolders (*e.g.,* one subfolder per participant). However, each gesture sample must be defined in its own JSON file in a consistent format.
- **Format.** A gesture sample is described by its class (*name*), the name or id of the user who performed it (*subject*), its *date* of creation, and a series of *paths*, *i.e.,* a path name and its associated strokes (*e.g.,* a uni-touch 2D gesture is described by one path, usually the index or the stylus, whereas a multi-path 3D hand gesture is described by a series of paths, one per path returned by the hand tracker). A stroke is an array of time-ordered points described by a set of coordinates (x, y, z, α, β, γ) and its *stroke id*.

3.3 Experimental Variables

The two main variables in Gester, like any other scientific experiment, are the independent (that are changed and controlled by the experimenter to test performance) and the dependent variables (that are tested and measured), which are defined in the next two subsections.

3.3.1 Independent Variables. Before executing the performance testing, the following variables need to be assigned to one or many values:

(1) Dataset1: nominal variable representing the first input dataset, mainly for individual testing.
(2) Dataset2: nominal variable representing the second input dataset, mainly for comparative testing.
(3) Number of resampling points (N): numerical variable with all conditions representing the number of points to which all gestures are resampled before classification. For example N={4, 8, 16}.
(4) Number of Templates (T): numerical variable with all conditions representing the number of templates per gesture class used for training. For example, T={32, 64}.
(5) Number of participants (P): numerical variable with all values representing the number of participants. For example, P={10, 20, 30}. Writing P=10+20+30 enables recognizers to be evaluated for the specific values P=10, P=20, and P=30 while writing P=15−20+30 evaluates for 15, 16, 17, 18, 19, 20, and 30.
(6) Number of repetitions (R): numerical variables representing the number of times that a test trial should be repeated for each gesture class. For example, R=100.

3.3.2 Dependent Variables. At the end of the evaluation process, Gester downloads an archive composed of CSV files divided into multiple subfolders (Figure 1). Gester creates one subfolder for each evaluation procedure, and each subfolder consists of:

- Confusion matrices. A confusion matrix is useful to practitioners for understanding which gesture classes are most confused with which other classes. Gester creates one set of confusion matrices per recognizer for all conditions, *i.e.,* all values of T templates per class and P independent participants. One global matrix reports results for all participants, whereas the remaining N matrices report results for each participant individually when P participants are considered. A row represents an actual class while a column represents the recognition rate of a predicted class. A recognizer should aim at maximizing the recognition rates displayed in the diagonal of the confusion matrix, *i.e.,* when actual classes match predicted classes.
- Rates per class. The recognition rates achieved by each recognizer for each class are also displayed in a separate file called *rates_class.csv*. The rates are reported for all conditions at once, *i.e.,* all values of T and P. A row represents an actual class while a column represents the recognition rate achieved by one specific recognizer.
- Rates per participant. The recognition rates achieved by each recognizer with each participant are displayed in a separate file called *rates_participant.csv*. The rates are reported for all conditions at once, *i.e.,* all values of T and P, when samples from the participant are used for testing. In this case, a row represents a pair of one participant and one recognizer (*e.g.,* if the dataset contains samples from 20 participants, Gester displays 20 rows, one per participant, for the first recognizer, then 20 rows for the second one, and so on for all recognizers).
- Recognition Rates. The recognition rates achieved by each recognizer in each condition are displayed in a separate file called *rates_global.csv*. A row represents a recognizer while a column represents the recognition rate achieved in one condition.
- Training times. The training times are reported for each recognizer in a separate file called *training_times.csv*. Training time corresponds to the time required to train the recognizer (*e.g.,* to pre-process all templates or to generate the classification model from all templates). A row represents a combination of

one testing participant and one recognizer, while a column represents the average training time. All times are in milliseconds (ms).

- Pre-processing time. The pre-processing times are reported for each recognizer in a separate file called *preprocessing_times.csv*. Pre-processing time is the time required to pre-process a candidate. A row represents a combination of one testing participant and one recognizer while a column is the pre-processing time averaged for all involved testing participants.
- Classification time. The classification times are reported for each recognizer in a separate file called *classification_times.csv*. Classification time corresponds to the time required to classify a pre-processed candidate (*e.g.,* to compare it to all pre-processed templates). A row represents a combination of one testing participant and one recognizer while a column represents the average classification time.
- Recognition time. The recognition times are reported for each recognizer in a separate file called *recognition_times.csv*. The recognition time is the sum of the pre-processing and the classification times for each candidate. A row represents a combination of one testing participant and one recognizer while a column represents the average recognition time.

4 Evaluation Procedures

Interactive applications require fast and highly accurate recognizers [46, 63]. The evaluation of state-of-art recognizers consists of assessing their accuracy and speed by the mean of measures, such as recognition rate and execution time. The evaluation of accuracy and speed requires (1) a *procedure*, *i.e.,* a set of parameters that define a particular performance testing scenario, and (2) a *dataset*, *i.e.,* a series of gesture samples that can be divided into a testing and a training set.

K-fold cross-validation [69] is a well-known procedure to evaluate the performance of a Machine Learning model [29]. The procedure consists of splitting a dataset into k groups such that each group is considered as a test set, or validation set, during the evaluation process. For each test set, accuracy and speed are estimated after training the model with the remaining groups. There exist two common procedures inspired by k-fold cross-validation among the studied recognizers, *i.e.,* user-dependent and user-independent [80]. Contrary to the original cross-validation, these procedures do not split a dataset into k groups: training and testing samples are randomly chosen within the entire dataset. We therefore define the procedures (or scenarios) as follows:

- **User-dependent procedure.** This evaluation consists of training and testing a recognizer with gestures produced by the same end user. User-dependent evaluation introduces two ordinal variables: (1) T, which designates the number of samples (Templates) used to train the recognizer, and (2) R, which designates the number of tests run (Repetitions) for each participant. The procedure works as follows:

(1) For each participant, T samples per gesture class are selected randomly for training, and one remaining sample per gesture class is selected randomly for testing.
(2) The process is repeated R times and the results are averaged to measure accuracy, *i.e.,* recognition rates, and speed, *i.e.,* average execution time.

▹ Recognizers evaluated in the user-dependent procedure found are: Rubine [58], xstroke [88], $1 [87], Protractor [44], $N [3], $P [80], Gestimator [93], Penny Pincher [70, 71], $P+ [78], $Q [82], and !FTL [75].

- **User-independent procedure.** This evaluation consists of training the recognizers with gestures produced by one or multiple end-users and testing with gestures produced by one (independent) end-user. User-independent evaluation introduces a third ordinal variable along with T and R: P, which designates the number of Participants used to train the recognizer. The procedure runs as follows:

(1) For each participant, one sample per gesture class is selected randomly for testing, and a set of P independent participants are chosen randomly to train the recognizer.

(2) For each participant in the training set, T samples per gesture class are selected randomly for training. Therefore, the recognizers are trained by $T \times P$ templates per class.
(3) The process is repeated R times and the results are averaged to measure accuracy, *i.e.,* recognition rates, and speed, *i.e.,* average execution time.
   ▹ Recognizers evaluated in the user-independent procedure found in the literature are: $P [80], SED-NE [95], Gestimator [93], Penny Pincher [70], $P+ [78], Jackknife [72], G-Gene [13], and $Q [82].

We just described the user-dependent and the user-independent procedures. Their main difference lies in the fact that gesture samples used for testing and training the recognizers are chosen from the same end-user in one case (user-dependent) and from different end-users in the other case (user-independent). Gester extends these procedures with *dataset-independence*: when recognizers are trained with samples from a first dataset and tested with samples from a second (independent) dataset. For example, the $1 dataset [87] contains gesture samples produced with three speeds, *i.e.,* slow, medium, and fast. Therefore, it can be divided into three subsets, one per speed, and we can evaluate the accuracy of all recognizers when they are trained by one subset and tested by another subset. Therefore, we are able to measure the impact of speed, input modality, and visual impairment on accuracy using this dataset-independent procedure. By combining user-(in)dependence and dataset-(in)dependence, Gester supports four evaluation procedures that are repeated R times to compute their average recognition rates and execution times:

- **User-dependent dataset-dependent procedure.** This evaluation consists of training and testing a recognizer with gestures produced by the same end-user from the same dataset. User-dependent evaluation introduces two ordinal variables: (1) T, which designates the number of training samples (templates) used to train the recognizer, and (2) R, which designates the number of tests executed for each participant. The procedure running is:
(1) For each participant, T samples per gesture class are selected randomly for training, and one remaining sample per gesture class is selected randomly for testing. All samples are chosen from the same dataset.
- **User-independent dataset-dependent procedure.** This evaluation consists of training the recognizers with gestures produced by one or multiple end-users from one dataset, and testing with gestures produced by one (independent) end-user from the same dataset. User-independent evaluation introduces a third ordinal variable called P, which designates the number of independent participants used to train the recognizer. The procedure runs as follows:
(1) For each participant, a set of P independent participants is randomly chosen to train the recognizer, and one sample per gesture class is randomly selected for testing. All samples are chosen from the same dataset.
(2) For each participant in the training set, T samples per gesture class are selected randomly for training within the same dataset.
- **User-dependent dataset-independent procedure.** This evaluation consists of training and testing a recognizer with gestures produced by one end-user from two different datasets (or from the same dataset divided into two subsets). The procedure runs as follows:
(1) For each participant in the first dataset, T samples per gesture class are selected randomly for training.
(2) For each participant in the second dataset, one remaining sample per class is selected randomly for testing.
- **User-independent dataset-independent procedure.** This evaluation consists of training the recognizers with gestures produced by one or multiple end-users from a first dataset and testing with gestures produced by one (independent) end-user within a second dataset. The procedure runs as follows:
(1) For each participant within the second dataset, one sample per gesture class is randomly selected for testing, and a set of P participants from the first dataset are randomly chosen for training.

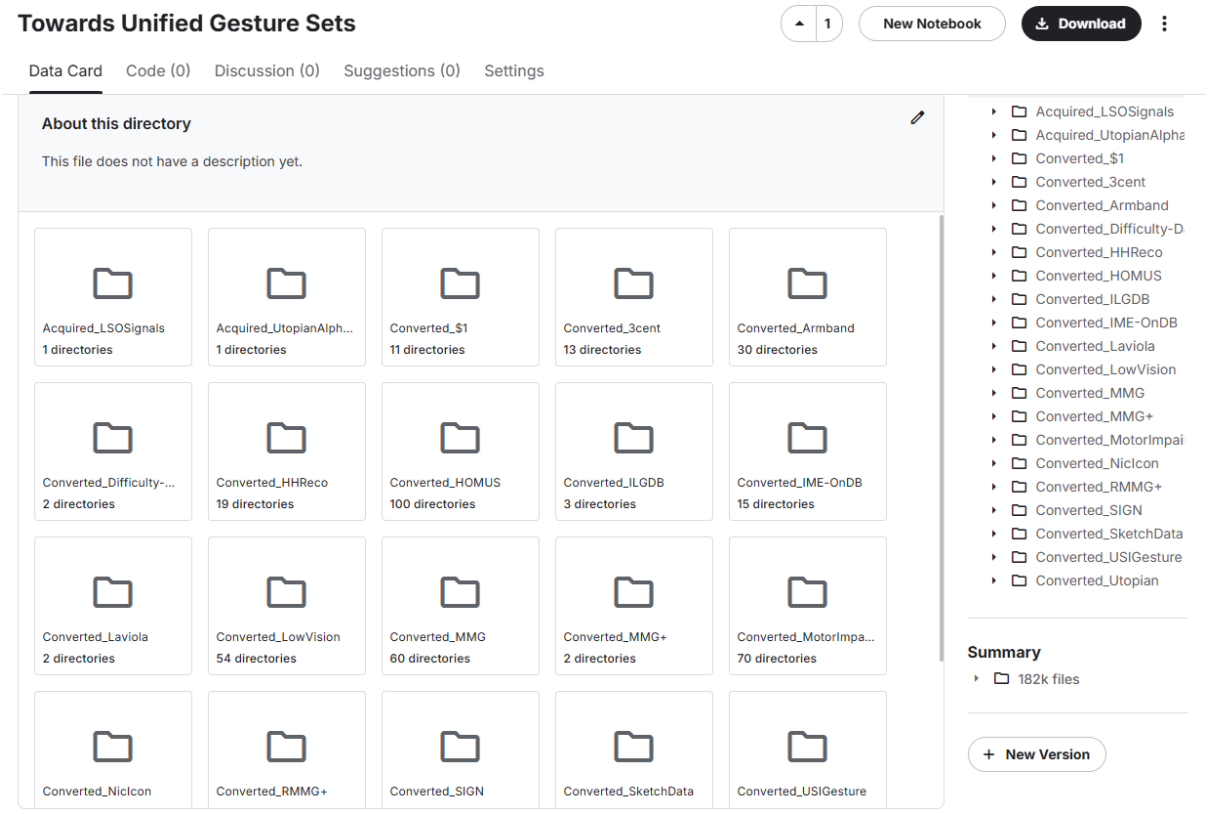


Fig. 4. Publicly available gesture sets in the unified gesture format in Kaggle.

(2) For each participant in the training set, T samples per gesture class are selected randomly for training. The samples are chosen from the first dataset.

In sum, the principle to decide a particular procedure is as follows: a user-dependent dataset-dependent test is launched if T, R, N, and Dataset1 are configured, a user-dependent dataset-independent test is run if T, R, N, Dataset1, and Dataset2 are configured, a user-independent dataset-dependent test is run if T, P, R, N, and Dataset1 configured, and a user-independent dataset-independent test is run if T, P, R, N, Dataset1, and Dataset2 are configured.

The MMG+ dataset [47] is divided into two subsets as 10 samples are recorded per participant for each gesture class: 5 on a smartphone and 5 on a tablet. By providing a smartphone dataset and a tablet dataset, we are entitled to run the user-(in)dependent dataset-(in)dependent tests. A data set can be split into two once the participants have created samples of all gesture classes in two or more categories (*e.g.,* speed in MMG [4], speed and device in MMG+ [47]). Since MMG has divided its set of participants into two subsets, one relying on fingers, and another relying on a stylus, we cannot perform user-dependent modality-independent as the participants are different. We can only apply the user-independent modality-independent procedure.

5 Engineering Activities Supported by Gester

This section illustrates typical activities (partially or totally) supported by Gester.

5.1 Existing Activities

5.1.1 Inserting a new dataset. This activity aims to insert a new dataset in Gester either by acquiring it directly according to the structure defined in subsection 3.2 or by converting an external dataset in this format. To this end, we developed a suite of JSON converters to obtain 21 unified gesture sets that are publicly available on Kaggle (Fig. 4). [9]

5.1.2 Inserting a new recognizer. . This activity aims to expand the scope of performance testing by inserting a new recognizer in two ways: by direct inclusion in the Javascript code or by an indirect method. For the first case, we inserted $C, a novel multi-stroke recognizer (see Appendix A for its formula [45]) based on Clifford algebra [14] to preserve the properties of invariance on-demand. For the second case, we considered $B, a recently developed gesture recognizer for gestures expressed via biosignals [90]. Python, the language in which $B has been implemented, cannot be inserted in JavaScript. Therefore, an indirect method should incorporate this recognizer, such as TransCrypt[10].

Table 1. Confusion matrix obtained by Gester testing the performance of !FTL on the $1 gesture set.

	Arrowhead	Asterisk	D letter	Excl. point	Five-point star	H letter	Half note	I letter	Line	N letter	Null	P letter	Pitchfork	Six-point star	T letter	X letter
Arrowhead	76.04	0.1	0.14	0.05	1.7	0.01	11.86		0.16	0.26	3.81	2.35	2.96		0.2	0.35
Asterisk	0.54	95.03		0.1	0.01	0.05				0.01	2.74			1.53		
D letter	1.08		89.41		0.16	0.33	0.04	0.01		0.45	1.45	4.55	0.04	2.49		
Excl. point				92.5					7.5							
Five-point star	0.24		0.01		97.35				0.01	1.05	0.18	0.06	1.1			
H letter	0.05	0.13	0.63		0.36	72.44		5.56		14.55	0.33	5.58		0.25	0.05	0.01
Half note	9.19	0.06		0.01			85.2	0.04	1.35	0.1	1.03	1.53	0.78		0.51	0.21
I letter			0.54			4.39	0.04	86.03	0.08	5.64	0.01	0.16		3.11	0.01	
Line			0.25						99.75							
N letter	0.74		0.7		0.33	7.84	0.08	4.4		82.85	0.88	1.98	0.01	0.2	0.01	
Null	3.25	0.24	0.28		2.18	0.04	0.69		0.3		89.31	1.51	0.76	1.45		
P letter	2.85	0.14	3.85		0.94	2.1	0.95		1.05	2.73		83.2	1.2	0.91		0.09
Pitchfork	4.3	3.54	0.13		0.79	0.04	3.24		0.05	0.01	0.63	0.54	83.39		0.26	3.1
Six-point star			0.19		1.85					0.18	0.11			97.68		
T letter	0.05				0.01	0.2	0.48		0.05			0.03			98.48	0.71
X letter	0.14	0.56				0.09	0.01				2.48		0.21			96.51

5.1.3 Testing the performance of a recognizer on a single gesture set. This activity aims to analyze the recognizer behavior on any class of the gesture set, which can be achieved class by class based on the measure and the confusion matrix and wheel. Concerning speed, we can detect which gesture classes are the fastest or the slowest recognized. Concerning accuracy, we can detect which gesture classes are best recognized, which are least well recognized, and which pairs of classes cause the most confusion. For example, Table 1 shows the confusion matrix obtained by Gester testing the user-independent, dataset-independent procedure on !FTL [75] with the following parameters: T=9600 , P=20, R=100, N=32, Dataset1=$1 [4]. The line, the T letter and the X letter were the best recognized, while the arrowhead and the H letter were the least well recognized. The H letter was confused with the N letter.

5.1.4 Testing the performance of a given recognizer on multiple gesture sets. This activity aims to determine which gesture set is recognized the best by a given recognizer, but also to what extent the recognizer is robust across diverse input conditions represented by various gesture sets. This activity typically includes the use of predefined gesture sets that contain a variety of samples that represent different styles [36], complexities [83], and

[9] See https://www.kaggle.com/datasets/jeanvanderdonckt/towards-unified-gesture-sets.
[10] See https://www.transcrypt.org/.

Table 2. Comparative testing of 9 recognizers by Gester on $1 dataset with T and P templates resampled to N=8 points.

Recognizer	T=1 –	T=2 –	T=4 –	T=8 –	T=1 P=1	T=2 P=2	T=4 P=4	T=8 P=8
Rubine	0.00	83.68	96.22	97.42	0.00	84.31	92.20	94.07
Protractor	94.72	97.66	98.65	99.27	85.52	93.91	97.69	99.22
$N-Pro	82.35	88.55	91.52	93.59	66.76	78.51	85.74	91.07
$P	89.95	95.23	97.51	98.75	74.71	88.40	95.39	98.34
Penny Pincher	93.61	96.64	98.07	98.92	86.13	93.86	97.59	98.93
$P+	95.89	98.54	99.22	99.65	89.16	95.72	98.51	99.36
JED	95.90	98.10	98.80	99.30	89.24	94.19	97.24	99.00
JIP	93.69	95.98	97.07	98.00	88.40	93.10	95.86	97.86
$Q	91.70	96.32	98.19	99.17	77.94	90.33	96.31	98.73
NFTL	71.86	80.38	86.30	90.23	59.59	74.65	83.69	90.55

articulations [79]. Cross-validation [69] with multiple gesture sets ensures generalizability, while stress testing with edge cases, such as overlapping gestures or incomplete input, highlights potential weaknesses of the tested recognizer. By analyzing the results, the recognizer can be fine-tuned to improve its robustness across diverse conditions. For a given recognizer, the best gesture set is found. The following activity provides the opposite answer: for a given gesture set, we look for the best recognizer under varying conditions.

5.1.5 Comparing the performance of two or more recognizers on a single gesture set. This activity aims to identify which recognizer performs the best in terms of accuracy and/or speed for a given gesture set. It is an instantiation of *comparative testing* [54] (also called *competitive testing*), an activity aimed at evaluating the strengths and weaknesses of two or more recognizers based on an existing dataset. Comparative testing should help gauge how well some recognizers perform in some equivalent conditions to identify areas for improvement. For example, Table 2 shows the results obtained by Gester testing ten recognizers with the following parameters: T=1+2+4+8, P=1+2+4+8, R=100, N=8, Dataset1=$1. Table 2 illustrates how the accuracy of all recognizers improves as the number of templates increases (*e.g.,* from 94.72% to 99:27% for Protractor [44]) because more templates cover more variations in gesture articulation. To account for the increase in accuracy from the smallest to the largest condition, Table 2 reports results for two user-dependent conditions, denoted as small (T=2) and large (T=8), and three user-independent conditions, denoted as small (T=2, P=2), large (T=4, P=4), and very large (T=8, P=8).

5.1.6 Thresholding recognizers on a single gesture set. This activity aims to verify that one or many recognizers achieve some threshold conditions regarding their accuracy and/or speed. For example, Table 3 shows the results of thresholding of three recognizers by Gester (*i.e.,* $P [80], $Q [82], and $C, which was introduced in the previous activity - see also Appendix A) with the following parameters: T=1+4, P=24, R=100, N=8, Dataset1=MMG. The threshold conditions imposed on accuracy and speed are $\tau \geq 90\%$ and $v \leq 1$ msec, respectively.

5.1.7 Performing a benchmarking of recognizers. This activity aims to compare a set of recognizers on one or multiple gesture sets to come up with a benchmark against which a new recognizer can be tested and compared [48]. Benchmarking should establish baseline values for a recognizer, setting the bar against which to compare and measure the improvements in future releases. Benchmarking performs a full comparative testing of existing *vs.* newly released recognizers to define thresholds. For example, let us benchmark $C, our novel recognizer introduced in this technical note, against two state-of-the-art recognizers, *i.e.,* $P [80] and $Q [82], on the Mixed Multistroke Gesture (MMG) symbol set [4]. Table 4 shows the results of this benchmarking using

Table 3. Thresholding of three recognizers by Gester: $P, $Q, and $C. The values of the parameters are: T=1+4, P=24, R=100, N=8. The subsets of MMG [4] considered are indicated in each configuration: MMG-Sm= smartphone, MMG-Tab= tablet, MMG-Sm-Tab= smartphone used for training, tablet used for testing, MMG-Tab-Sm= tablet used for training, smartphone used for testing. I denotes the number of independent participants used for training. The threshold for accuracy is set to ≥ 90% and ≤ 1 msec for speed. Green cells satisfy the threshold condition, and red cells do not.

		$P		$Q		$C	
Configurations	Units	T=1	T=4	T=1	T=4	T=1	T=4
User-dependent, dataset-dependent	%	86.69	95.71	88.19	96.97	83.94	93.68
MMG-Sm	ms	0.08	0.28	0.36	0.38	0.10	0.29
MMG-Tab	%	88.40	96.20	89.51	96.89	85.89	95.29
	ms	0.08	0.28	0.32	0.37	0.10	0.28
User-independent, dataset-dependent	%	63.25	88.85	60.84	90.31	66.89	87.34
MMG-Sm, I=1..5	ms	0.08	1.42	0.32	0.69	0.10	1.37
MMG-Tab, I=1..5	%	66.36	90.15	64.76	90.90	68.71	89.37
	ms	0.08	1.42	0.33	0.68	0.10	1.34
User-dependent, dataset-independent	%	83.49	93.05	83.85	93.50	81.17	90.59
MMG-Sm-Tab	ms	0.08	0.28	0.32	0.37	0.10	0.30
MMG-Tab-Sm	%	83.73	92.07	83.57	92.65	80.28	89.45
	ms	0.08	0.37	0.32	0.37	0.10	0.30
User-independent, dataset-independent	%	64.53	89.37	62.27	90.42	67.71	89.27
MMG-Sm-Tab	ms	0.08	1.42	0.32	0.69	0.10	1.37
MMG-Tab-Sm	%	64.33	88.35	61.45	90.45	67.47	87.59
	ms	0.08	1.42	0.32	0.69	0.10	1.37

Gester in the homogeneous conditions of the four procedures defined in section 4 with the following parameters: T=1+4, P=24, R=100, N=8. We define a benchmarking based on the admissible intervals of $[80, 88]$%, $[89, 94]$% for the accuracy [61], and $[0.1, 0.3]$ msec, $[0.1, 0.4]$ msec, for T=1 and T=4, respectively. In the user-dependent, dataset-independent procedure, Table 4 indicates that all three recognizers benefit from admissible accuracies (their cells are either blank -within the interval- or above - in green). $Q has a speed exceeding the interval $[0.1, 0.3]$ msec for T=1 (two cells are in red with a value of 0.36 msec and 0.32 msec, respectively). In this configuration, we observe that our new recognizer $C achieves an accuracy of 93.68% (T=4), which is within the admissible interval, but lower than $P (95.71%) and $Q (96.97%), the best recognition rate in all conditions. This result can be put into perspective with other results obtained using other algorithms on the same dataset, such as 89.96% with a Convolutional Neural Network [18]. As these results were obtained outside Gester, their experimental conditions are not entirely comparable. Similar conclusions can be drawn by observing the table.

5.1.8 Testing the performance of a new recognizer. This activity aims to test the performance of a newly introduced recognizer (after insertion - see above). Once a new recognizer is inserted in Gester, its performance can be immediately tested. For example, we inserted and tested the performance of PolyRec [26], which is the stroke gesture recognizer incorporated in PolyRec Gesture Design Tool [11]. We selected PolyRec because its published results show the potential to outperform alternative recognizers, at least for unistroke gesture datasets.

Table 4. Benchmarking of three recognizers by Gester: $P, $Q, and $C. The values of the parameters are: T=1+4, P=24, R=100, N=8. The subsets of MMG [4] considered are indicated in each configuration: MMG-Sm= smartphone, MMG-Tab= tablet, MMG-Sm-Tab= smartphone used for training, tablet used for testing, MMG-Tab-Sm= tablet used for training, smartphone used for testing. I denotes the number of independent participants used for training. Red and green cells depict values below the minimal threshold, above the maximal threshold defined in the benchmarking, which depends on T, respectively. Uncolored cells depict admissible values.

		$P		$Q		$C	
Configurations	Units	T=1	T=4	T=1	T=4	T=1	T=4
User-dependent, dataset-dependent	%	86.69	95.71	88.19	96.97	83.94	93.68
MMG-Sm	ms	0.08	0.28	0.36	0.38	0.10	0.29
MMG-Tab	%	88.40	96.20	89.51	96.89	85.89	95.29
	ms	0.08	0.28	0.32	0.37	0.10	0.28
User-independent, dataset-dependent	%	63.25	88.85	60.84	90.31	66.89	87.34
MMG-Sm, I=1..5	ms	0.08	1.42	0.32	0.69	0.10	1.37
MMG-Tab, I=1..5	%	66.36	90.15	64.76	90.90	68.71	89.37
	ms	0.08	1.42	0.33	0.68	0.10	1.34
User-dependent, dataset-independent	%	83.49	93.05	83.85	93.50	81.17	90.59
MMG-Sm-Tab	ms	0.08	0.28	0.32	0.37	0.10	0.30
MMG-Tab-Sm	%	83.73	92.07	83.57	92.65	80.28	89.45
	ms	0.08	0.37	0.32	0.37	0.10	0.30
User-independent, dataset-independent	%	64.53	89.37	62.27	90.42	67.71	89.27
MMG-Sm-Tab	ms	0.08	1.42	0.32	0.69	0.10	1.37
MMG-Tab-Sm	%	64.33	88.35	61.45	90.45	67.47	87.59
	ms	0.08	1.42	0.32	0.69	0.10	1.37

This may in part be due to its preprocessing strategy, where strokes are resampled along their dominant points, which has the potential to preserve important low-frequency information that can be lost with uniform spatial resampling, the strategy used by other recognizers [47]. We tested the performance of PolyRec using Gester in user-dependent and dataset-dependent and user-independent and dataset-dependent procedures with the following parameters: T=2+4+8+16, P=24, R=100, N=8+16, Dataset1=MMG. We provide in supplementary material the complete results of this testing as follows:

- `confusion_matrices.csv`: one confusion matrix per participant, the first row of each confusion matrix is "participant;class1;class2;...;classN;No match".
- `classification_times.csv`, where the columns represent the number of training templates, from smallest to largest, and the rows represent the participant.
- `preprocessing_times.csv`, where the columns represent the number of training templates, from smallest to largest, and the rows represent the participant.
- `rates_class.csv`: average recognition rate for each gesture class for all conditions where the rows represent the gesture.

- `rates_global.csv`: average recognition rate for each number of training templates, where the columns represent the number of training templates, from smallest to largest.
- `rates_participant.csv`, where the columns represent the number of training templates, from smallest to largest, and the rows represent the participant.
- `recognition_times.csv`, where the columns represent the number of training templates, from smallest to largest, and the rows represent the participant.
- `training_times.csv`, where the columns represent the number of training templates, from smallest to largest, and the rows represent the participant.

5.1.9 Repeat the performance testing of recognizers. This activity aims to repeat previously conducted performance testing to check whether these results remain consistent under the same conditions (*i.e.,* same experimenters or testers, same experimental setup). For example, benchmarking can be repeated or updated to prove that an experimenter can reliably repeat her testing.

5.1.10 Replicate the performance testing of recognizers. This activity aims to replicate a previously conducted performance testing to check that an independent experimenter can obtain the same result using the initial experimenter's setup (*i.e.,* different experimenters or testers, same experimental setup).

5.1.11 Reproduce the performance testing of recognizers. This activity aims to reproduce a previously conducted performance testing to investigate how the results would change under different conditions (*i.e.,* different experimenters, different experimental setups). Testing varying conditions is crucial for ensuring the robustness, accuracy, and generalizability of the recognizer as it often encounters diverse users, platforms (*e.g.,* a more powerful computer, upgraded computational capabilities or constraints [92]), and environments. Testing under these conditions allows developers to identify weaknesses and improve the recognizer's ability to perform reliably in other scenarios, perhaps unforeseen.

5.1.12 Repurpose the performance comparison of recognizers. This activity aims to reproduce previously performed performance test to pursue goals other than those initially planned [27]. Performance testing can be repurposed for goals beyond their initial intent, unlocking new insights and gesture applications. For example, gestures considered during performance testing can be analyzed to understand user behavior, preferences, and ergonomics [89], aiding in the design of more usable gesture-based UIs or devices. Furthermore, the testing process can reveal patterns or anomalies that contribute to broader gesture recognition research, such as improving algorithms for gesture prediction or estimation [41, 42]. Gesture recognizer performance testing can also be adapted to evaluate accessibility features, ensuring inclusivity for users with physical or cognitive impairments. The insights gained can support related fields like virtual reality, gaming, or rehabilitation, where gesture-based UIs are integral, enabling innovative applications that transcend the original purpose of the recognizer.

5.2 Limitations of Gester

Most of the case studies discussed above are feasible, provided that the necessary conditions for their execution are met, which may be conditioned by the following limitations:

- **Limitations imposed by the JavaScript implementation**. Gester is implemented in JavaScript, one the most used programming languages motivated by its support of cross-platform development, for both server-side and client-side applications. JavaScript also benefits from an active community of developers, making it easy to incorporate new recognizers and procedures in Gester. Moreover, JavaScript today plays a more significant role in emerging technologies involving devices and sensors used for gestural interaction. However, JavaScript's single inheritance model allows objects to inherit properties and behaviors from a single prototype object, which forces us to declare the testing variables, such as N, P, R, and the variables

specific to a recognizer as global variables. To incorporate a recognizer implemented in another language, such as $B [90] in Python, some tricky mechanisms should be used, such as indirect methods. Moreover, JavaScript is prone to inherent performance limitations, such as memory limitations (*e.g.,* we tested a configuration that resulted in memory overload and was never completed; moving on to a computer equipped with more RAM did not solve that problem) and computational constraints (*e.g.,* we tested a configuration that required two weeks on a high-end laptop to be completed!), which may require appropriate classification models [92]. One major limitation of Gester due to its implementation in JavaScript is its single-threaded nature, which can lead to performance issues when executing complex tasks that are heavily computational. Furthermore, its weakly typed nature can result in unintended type coercion, leading to unpredictable behavior. JavaScript's reliance on the browser environment means inconsistencies across different browsers, requiring extra effort for cross-browser compatibility. Lastly, its asynchronous execution model, while beneficial for responsiveness, can introduce callback problems.

- **Limitations imposed by recognizers and datasets**. The performance of recognizers can be tested on accessible, reference, and challenging datasets provided that they are publicly available and they can be incorporated in Gester, despite their capabilities. To transform the data structure of a publicly available gesture set into the unified format manipulated by Gester, manual coding using XSLT transformations may be required, although a more assisted manner could rely on a visual transformation platform, such as Babelway[11] via some conversion protocol that can be reused and easily modified. Up to now, offline recognizers are integrated that require that the entire gesture traces be pre-recorded, therefore discarding interesting online recognizers that enable real-time gesture recognition while the end-user or the writer is issuing a gesture, such as G-Gene [13] or declarative-based approaches [20]. Gester does not exploit the capabilities offered by initiatives to synthesize [6] new gestures from existing templates, such as by synthetic data generation [70], by data augmentation techniques [51], by bootstrapping [41, 50], or to obtain templates collected by the crowd [34].
- **Limitations imposed by testing procedures**. The Leave-one-out cross-validation (LOOCV), a modified form of the k-fold cross-validation [69] with k=1, was used for the user-independent scenario. Increasing the value k should expand the coverage and the guarantee to obtain more reliable results, but the number of combinations grows exponentially, such as 7 tests for k=3 instead of one test for k=1. Gester could be extended by implementing other cross-validation procedures[12], ranging from the simple, repeated, or stratified k-fold techniques to Leave-One-Out (LOO) or Leave-P-Out (LPO) and Shuffle and Split techniques. For example, GestureVlad [7] tested the recognition performance using LOOCV and split method, such as 50%-50%. Other repartitions, such as 20%-80% suggested by the Pareto principle [10] and 30%-70% can be imagined as well, although ye do not know the effect of this repartition on performance testing.
  Our procedures are aimed at testing the "system" performance of recognizers, which do not compute other measures that test or estimate the "human" performance, such as the visual similarity [35], their consistency [2], the measures for human performance [42], the shape of gestures [12, 45], and their perceived difficulty [83], and their measures for relative accuracy [81]. Beyond these measures that estimate or predict the system and the human performance, recognizers should be tested ultimately in real-world running applications based on usability testing to confirm or disconfirm the results [67]. The procedures should remain systematic [54], which requires its parameters be properly declared and have an appropriate range of values to be determined explicitly.

[11] See https://www.babelway.com/.
[12] See for example https://scikit-learn.org/stable/modules/cross_validation.html

6 Conclusion and Future Work

We present Gester, an offline software that automates the performance testing of stroke gesture recognizers in terms of three recognition rates for assessing accuracy and four execution times to assess the speed. It is implemented in HTML and Javascript, two languages suited for rapid prototyping of gesture-based UIs of interactive applications [3, 70]. Therefore, it expects that all recognizers are implemented uniformly in Javascript and all datasets are imported in the consistent JSON format used by Gester to preserve the homogeneity of experimental conditions. Gester supports four evaluation procedures, from user-dependent dataset-dependent to user-independent dataset-independent, to simulate as many use case scenarios as possible for evaluating their system performance, which should be distinguished from the human performance [42]. The application returns results expressed in terms of recognition rates for accuracy, times for speed, and confusion matrices to understand classification mistakes. These results are downloaded directly on the end-user's device in separate CSV files that are easy-to-import in most statistical programs or a model explorer [73]. All Chromium-based browsers support the loading of files from one folder and its subfolders.

Therefore, Gester must be used specifically on a browser of this family for compatibility. Adding a new recognizer to the pool of already implemented recognizers or a new dataset is a matter of directory inclusion (no need to retrain or modify the code). Still, JavaScript should be always used. Selecting the most performant recognizer for a given gesture set to be used in a given context of use remains a complex problem [60] that would invite a new procedure that systematically tests a gesture set against all configurations of supported recognizers.

Future work will consider *ensemble learning* [57], a machine learning technique that aggregates two or more recognizers to produce a better performance, specifically a better accuracy, than relying on a single recognizer. The literature categorizes ensemble learning methods into two groups:

- *Parallel methods*, which train each base recognizer apart from the others in a parallel and independent fashion from one another;
- *Sequential methods*, which train a new base recognizer to minimize errors made by the previous recognizer trained in the preceding step to construct base models sequentially in stages.

Another option for future work consists of inventing an *adaptive gesture recognizer* that would automatically select the best recognizer based on meta-learning [17] and show the adaptivity [19]. Up to now, implemented recognizers only provide one accuracy result at a time for each candidate gesture or gesture class, often in terms of a percentage or a value between 0 and 1 depending on the similarity measure. The recognition process could be further relaxed or constrained according to the criticality imposed by the context of use: the recognition of a gesture for a command [5] could benefit from a wider tolerance than that of a gesture used for end-user authentication [91] or a human signature [9].

Open Science. We share a Gester[13] repository with the software developed in this paper, all recognizers written in JavaScript and datasets converted into a unified JSON format to foster reproducible research around gesture recognition and its incorporation into mainstream interactive applications. The recognizer introduced is also publicly available on our μV repository.

Acknowledgments

Nathan Magrofuoco is funded by the UCLouvain Fonds Speciaux de Recherche under Grant No. 61273304 and by FRIA under Grant No. FC 31773. Jean Vanderdonckt is supported by the EU EIC Pathfinder-Awareness Inside challenge Symbiotik project (1 Oct. 2022-30 Sept. 2026) under Grant no. 101071147.

[13] See https://github.com/nmagrofuoco/Gester.

References

[1] Abdullah X. Ali, Meredith Ringel Morris, and Jacob O. Wobbrock. 2019. Crowdlicit: A System for Conducting Distributed End-User Elicitation and Identification Studies. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300485

[2] Lisa Anthony, Radu-Daniel Vatavu, and Jacob O. Wobbrock. 2013. Understanding the consistency of users' pen and finger stroke gesture articulation. In *Proc. of Graphics Interface 2013* (Regina, SK, Canada) *(GI '13)*, Faramarz F. Samavati and Kirstie Hawkey (Eds.). Canadian Information Processing Society / ACM, 87–94. http://dl.acm.org/citation.cfm?id=2532145

[3] Lisa Anthony and Jacob O. Wobbrock. 2010. A Lightweight Multistroke Recognizer for User Interface Prototypes. In *Proc. of Graphics Interface 2010* (Ottawa, Ontario, Canada) *(GI '10)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 245–252. http://dl.acm.org/citation.cfm?id=1839214.1839258

[4] Lisa Anthony and Jacob O. Wobbrock. 2012. $N-Protractor: A Fast and Accurate Multistroke Recognizer. In *Proc. of Graphics Interface 2012* (Toronto, Ontario, Canada) *(GI '12)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 117–120. http://dl.acm.org/citation.cfm?id=2305276.2305296

[5] Caroline Appert and Shumin Zhai. 2009. Using strokes as command shortcuts: cognitive benefits and toolkit support. In *Proc. of the 27th Int. Conf. on Human Factors in Computing Systems* (Boston, MA, USA) *(CHI '09)*, Dan R. Olsen Jr., Richard B. Arthur, Ken Hinckley, Meredith Ringel Morris, Scott E. Hudson, and Saul Greenberg (Eds.). Association for Computing Machinery, New York, NY, USA, 2289–2298. https://doi.org/10.1145/1518701.1519052

[6] Marvin Bachert and Marc Hesenius. 2024. Towards a Framework for Evaluating Synthetic Surface Gestures. In *Companion Proceedings of the 16th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Cagliari, Italy) *(EICS '24 Companion)*. Association for Computing Machinery, New York, NY, USA, 22–30. https://doi.org/10.1145/3660515.3661327

[7] Abel Díaz Berenguer, Meshia Cédric Oveneke, Habib-Ur-Rehman Khalid, Mitchel Alioscha-Perez, André Bourdoux, and Hichem Sahli. 2019. GestureVLAD: Combining Unsupervised Features Representation and Spatio-Temporal Aggregation for Doppler-Radar Gesture Recognition. *IEEE Access* 7 (2019), 137122–137135. https://doi.org/10.1109/ACCESS.2019.2942305

[8] François Beuvens and Jean Vanderdonckt. 2012. UsiGesture: An environment for integrating pen-based interaction in user interface development. In *Proc. of 6th International Conference on Research Challenges in Information Science* (Valencia, Spain) *(RCIS 2012)*, Colette Rolland, Jaelson Castro, and Oscar Pastor (Eds.). IEEE, 1–12. https://doi.org/10.1109/RCIS.2012.6240449

[9] Azra Izni Anis binti Azlin, Pang Ying Han, Khoh Wee How, and Ooi Shih Yin. 2023. Hand Gesture Signature Recognition with Machine Learning Algorithms. In *Proceedings of the 9th International Conference on Computational Science and Technology (Lecture Notes in Electrical Engineering, Vol. 983)*, Dae-Ki Kang, Rayner Alfred, Zamhar Iswandono Bin Awang Ismail, Aslina Baharum, and Vinesh Thiruchelvam (Eds.). Springer Nature Singapore, Singapore, 389–398. https://doi.org/10.1007/978-981-19-8406-8_30

[10] George E.P. Box and R. Daniel Meyer. 1986. An Analysis for Unreplicated Fractional Factorials. *Technometrics* 28, 1 (1986), 11–18. https://doi.org/10.1080/00401706.1986.10488093 arXiv:https://www.tandfonline.com/doi/pdf/10.1080/00401706.1986.10488093

[11] Roberto Bufano, Gennaro Costagliola, Mattia De Rosa, and Vittorio Fuccella. 2022. PolyRec Gesture Design Tool: A tool for fast prototyping of gesture-based mobile applications. *Software: Practice and Experience* 52, 2 (2022), 594–618. https://doi.org/10.1002/spe.3024 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3024

[12] Alexandre Calado, Paolo Roselli, Vito Errico, Nathan Magrofuoco, Jean Vanderdonckt, and Giovanni Saggio. 2022. A Geometric Model-Based Approach to Hand Gesture Recognition. *IEEE Trans. Syst. Man Cybern. Syst.* 52, 10 (2022), 6151–6161. https://doi.org/10.1109/TSMC.2021.3138589

[13] Alessandro Carcangiu and Lucio Davide Spano. 2018. G-Gene: A Gene Alignment Method for Online Partial Stroke Gestures Recognition. *Proc. ACM Hum.-Comput. Interact.* 2, EICS, Article 13 (June 2018), 17 pages. https://doi.org/10.1145/3229095

[14] William Kingdon Clifford. 1871. Preliminary Sketch of Biquaternions. *Proceedings of the London Mathematical Society* s1-4, 1 (1871), 381–395. https://doi.org/10.1112/plms/s1-4.1.381 arXiv:https://londmathsoc.onlinelibrary.wiley.com/doi/pdf/10.1112/plms/s1-4.1.381

[15] Gennaro Costagliola, Mattia De Rosa, and Vittorio Fuccella. 2023. *Gesture-Based Computing*. John Wiley & Sons, Ltd, Chapter 32, 397–408. https://doi.org/10.1002/9781119863663.ch32 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119863663.ch32

[16] Adrien Coyette, Sascha Schimke, Jean Vanderdonckt, and Claus Vielhauer. 2007. Trainable Sketch Recognizer for Graphical User Interface Design. In *Proc. of 11th IFIP TC 13 International Conference on Human-Computer Interaction, INTERACT '07* (Rio de Janeiro, Brazil) *(Lecture Notes in Computer Science, Vol. 4662)*, Maria Cecília Calani Baranauskas, Philippe A. Palanque, Julio Abascal, and Simone Diniz Junqueira Barbosa (Eds.). Springer, Berlin Heidelberg, 124–135. https://doi.org/10.1007/978-3-540-74796-3_14

[17] Itai Dagan, Roman Vainshtein, Gilad Katz, and Lior Rokach. 2024. Automated algorithm selection using meta-learning and pre-trained deep convolution neural networks. *Inf. Fusion* 105 (2024), 102210. https://doi.org/10.1016/J.INFFUS.2023.102210

[18] Quentin Debard, Christian Wolf, Stéphane Canu, and Julien Arné. 2018. Learning to recognize touch gestures: recurrent vs. convolutional features and dynamic sampling. *CoRR* abs/1802.09901 (2018). arXiv:1802.09901 http://arxiv.org/abs/1802.09901

[19] Charles-Eric Dessart, Vivian Genaro Motti, and Jean Vanderdonckt. 2011. Showing User Interface Adaptivity by Animated Transitions. In *Proceedings of the ACM Symposium on Engineering Interactive Computing Systems* (Pisa, Italy) *(EICS '11)*. Association for Computing

Machinery, New York, NY, USA, 95–104. https://doi.org/10.1145/1996461.1996501

[20] Stefano Dessì and Lucio Davide Spano. 2020. DG3: Exploiting Gesture Declarative Models for Sample Generation and Online Recognition. *Proc. ACM Hum.-Comput. Interact.* 4, EICS, Article 82 (June 2020), 21 pages. https://doi.org/10.1145/3397870

[21] Anind K. Dey. 2001. Understanding and Using Context. *Pers. Ubiquitous Comput.* 5, 1 (2001), 4–7. https://doi.org/10.1007/s007790170019

[22] Ana Belén Erazo and Jorge Luis Pérez Medina. 2020. Algorithmic Efficiency of Stroke Gesture Recognizers: a Comparative Analysis. *International Journal on Advanced Science, Engineering and Information Technology* 10, 2 (Mar. 2020), 438–446. https://doi.org/10.18517/ijaseit.10.2.10807

[23] Mark Fewster and Dorothy Graham. 1999. *Software test automation: effective use of test execution tools.* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. https://doi.org/doi/10.5555/312384

[24] Ivo Aluízio Stinghen Filho, Estevam Nicolas Chen, Jucimar Maia da Silva Junior, and Ricardo da Silva Barboza. 2018. Gesture recognition using leap motion: a comparison between machine learning algorithms. In *ACM SIGGRAPH 2018 Posters* (Vancouver, British Columbia, Canada) *(SIGGRAPH '18)*. Association for Computing Machinery, New York, NY, USA, Article 65, 2 pages. https://doi.org/10.1145/3230744.3230750

[25] Clive Frankish, Richard Hull, and Pam Morgan. 1995. Recognition Accuracy and User Acceptance of Pen Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '95)*. ACM Press/Addison-Wesley Publishing Co., USA, 503–510. https://doi.org/10.1145/223904.223972

[26] Vittorio Fuccella and Gennaro Costagliola. 2015. Unistroke Gesture Recognition Through Polyline Approximation and Alignment. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 3351–3354. https://doi.org/10.1145/2702123.2702505

[27] Bogdan-Florin Gheran, Santiago Villarreal-Narvaez, Radu-Daniel Vatavu, and Jean Vanderdonckt. 2022. RepliGES and GEStory: Visual Tools for Systematizing and Consolidating Knowledge on User-Defined Gestures. In *Proceedings of the International Conference on Advanced Visual Interfaces* (Frascati, Rome, Italy) *(AVI 2022)*, Paolo Bottoni and Emanuele Panizzi (Eds.). ACM, 5:1–5:9. https://doi.org/10.1145/3531073.3531112

[28] Nicholas Edward Gillian and Joseph A. Paradiso. 2014. The gesture recognition toolkit. *The Journal of Machine Learning Research* 15, 1 (2014), 3483–3487. https://doi.org/10.5555/2627435.2697076

[29] Juan M. Gorriz, F. Segovia, J. Ramirez, A. Ortiz, and J. Suckling. 2024. Is K-fold cross validation the best model selection method for Machine Learning? arXiv:2401.16407 [stat.ML] https://arxiv.org/abs/2401.16407

[30] Tobias Griebe, Marc Hesenius, and Volker Gruhn. 2015. Towards Automated UI-Tests for Sensor-Based Mobile Applications. In *Intelligent Software Methodologies, Tools and Techniques*, Hamido Fujita and Guido Guizzi (Eds.). Springer International Publishing, Cham, 3–17.

[31] Marc Hesenius, Tobias Griebe, Stefan Gries, and Volker Gruhn. 2014. Automating UI tests for mobile applications with formal gesture descriptions. In *Proceedings of the 16th International Conference on Human-Computer Interaction with Mobile Devices & Services* (Toronto, ON, Canada) *(MobileHCI '14)*. Association for Computing Machinery, New York, NY, USA, 213–222. https://doi.org/10.1145/2628363.2628391

[32] Lode Hoste and Beat Signer. 2014. Criteria, Challenges and Opportunities for Gesture Programming Languages. In *Proceedings of the Workshop on Engineering Gestures for Multimodal Interfaces Co-located with the 6th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EGMI@EICS 2014, Rome, Italy, June 17, 2014 (CEUR Workshop Proceedings, Vol. 1190)*, Florian Echtler, Lode Hoste, Dietrich Kammer, Beat Signer, and Davy Vanacken (Eds.). CEUR-WS.org, 22–29. https://ceur-ws.org/Vol-1190/paper4.pdf

[33] Heloise Hse, Michael Shilman, and A. Richard Newton. 2004. Robust Sketched Symbol Fragmentation Using Templates. In *Proc. of the 9th Int. Conf. on Intelligent User Interfaces* (Funchal, Madeira, Portugal) *(IUI '04)*. Association for Computing Machinery, New York, NY, USA, 156–160. https://doi.org/10.1145/964442.964472

[34] JuChan Seo In-Taek Jung, Sooyeon Ahn and Jin-Hyuk Hong. 2024. Exploring the Potentials of Crowdsourcing for Gesture Data Collection. *International Journal of Human–Computer Interaction* 40, 12 (2024), 3112–3121. https://doi.org/10.1080/10447318.2023.2180235 arXiv:https://doi.org/10.1080/10447318.2023.2180235

[35] Allan Christian Long Jr., James A. Landay, Lawrence A. Rowe, and Joseph Michiels. 2000. Visual similarity of pen gestures. In *Proc. of the ACM Int. Conf. on Human factors in computing systems* (The Hague, The Netherlands) *(CHI '00)*, Thea Turner and Gerd Szwillus (Eds.). Association for Computing Machinery, New York, NY, USA, 360–367. https://doi.org/10.1145/332040.332458

[36] Woojin Kang, In-Taek Jung, DaeHo Lee, and Jin-Hyuk Hong. 2021. Styling Words: A Simple and Natural Way to Increase Variability in Training Data Collection for Gesture Recognition. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 318, 12 pages. https://doi.org/10.1145/3411764.3445457

[37] Shahedul Huq Khandkar, S. M. Sohan, Jonathan Sillito, and Frank Maurer. 2010. Tool support for testing complex multi-touch gestures. In *ACM International Conference on Interactive Tabletops and Surfaces* (Saarbrücken, Germany) *(ITS '10)*. Association for Computing Machinery, New York, NY, USA, 59–68. https://doi.org/10.1145/1936652.1936663

[38] Mary LaLomia. 1994. User acceptance of handwritten recognition accuracy. In *Conference Companion on Human Factors in Computing Systems* (Boston, Massachusetts, USA) *(CHI '94)*. Association for Computing Machinery, New York, NY, USA, 107–108. https://doi.org/

10.1145/259963.260086

[39] Joseph J. LaViola Jr. and Robert C. Zeleznik. 2007. A Practical Approach for Writer-Dependent Symbol Recognition Using a Writer-Independent Symbol Recognizer. *IEEE Trans. Pattern Anal. Mach. Intell.* 29, 11 (Nov. 2007), 1917–1926. https://doi.org/10.1109/TPAMI.2007.1109

[40] Emanuele Ledda and Lucio Davide Spano. 2021. Applying Long-Short Term Memory Recurrent Neural Networks for Real-Time Stroke Recognition. In *Companion of the 2021 ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Virtual Event, Netherlands) *(EICS '21)*. Association for Computing Machinery, New York, NY, USA, 50–55. https://doi.org/10.1145/3459926.3464754

[41] Luis A. Leiva, Daniel Martín-Albo, and Réjean Plamondon. 2016. Gestures à Go Go: Authoring Synthetic Human-Like Stroke Gestures Using the Kinematic Theory of Rapid Movements. *ACM Trans. Intell. Syst. Technol.* 7, 2 (2016), 15:1–15:29. https://doi.org/10.1145/2799648

[42] Luis A. Leiva, Radu-Daniel Vatavu, Daniel Martín-Albo, and Réjean Plamondon. 2020. Omnis Prædictio: Estimating the full spectrum of human performance with stroke gestures. *International Journal of Human-Computer Studies* 142 (2020), 102466. https://doi.org/10.1016/j.ijhcs.2020.102466

[43] Panagiotis Leloudas. 2023. *Introduction to Software Testing - A Practical Guide to Testing, Design, Automation, and Execution.* Apress, Berkeley, CA, USA. https://doi.org/10.1007/978-1-4842-9514-4

[44] Yang Li. 2010. Protractor: A Fast and Accurate Gesture Recognizer. In *Proc. of the ACM Int. Conf. on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 2169–2172. https://doi.org/10.1145/1753326.1753654

[45] Lorenzo Luzzi and Paolo Roselli. 2020. The shape of planar smooth gestures and the convergence of a gesture recognizer. *Aequat. Math.* 94 (2020), 219–233. https://doi.org/10.1007/s00010-020-00712-7

[46] Nathan Magrofuoco, Paolo Roselli, and Jean Vanderdonckt. 2022. Two-dimensional Stroke Gesture Recognition: a Survey. *Comput. Surveys* 54, 7 (2022), 155:1–155:36. https://doi.org/10.1145/3465400

[47] Nathan Magrofuoco, Paolo Roselli, and Jean Vanderdonckt. 2022. μV: An Articulation, Rotation, Scaling, and Translation Invariant (ARST) Multi-stroke Gesture Recognizer. *Proc. ACM Hum.-Comput. Interact.* 6, EICS, Article 150 (June 2022), 25 pages. https://doi.org/10.1145/3532200

[48] Nathan Magrofuoco, Arthur Sluyters, Paolo Roselli, and Jean Vanderdonckt. 2025. Comparative Testing of 2D Stroke Gesture Recognizers in Multiple Contexts. *Proc. ACM Hum.-Comput. Interact.* 10, EICS, Article 6 (June 2025), 35 pages.

[49] Nathan Magrofuoco and Jean Vanderdonckt. 2019. Gelicit: A Cloud Platform for Distributed Gesture Elicitation Studies. *Proc. ACM Hum.-Comput. Interact.* 3, EICS, Article 6 (June 2019), 41 pages. https://doi.org/10.1145/3331148

[50] Daniel Martín-Albo and Luis A. Leiva. 2016. G3: bootstrapping stroke gestures design with synthetic samples and built-in recognizers. In *Proc. of the 18th Int. Conf. on Human-Computer Interaction with Mobile Devices and Services Adjunct* (Florence, Italy) *(MobileHCI '16)*, Fabio Paternò, Kaisa Väänänen, Karen Church, Jonna Häkkilä, Antonio Krüger, and Marcos Serrano (Eds.). Association for Computing Machinery, New York, NY, USA, 633–637. https://doi.org/10.1145/2957265.2961833

[51] Mykola Maslych, Eugene Matthew Taranta, Mostafa Aldilati, and Joseph J. Laviola. 2023. Effective 2D Stroke-Based Gesture Augmentation for RNNs. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 282, 13 pages. https://doi.org/10.1145/3544548.3581358

[52] International Standard Organization. 2011. ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. https://www.iso.org/standard/35733.html

[53] International Standard Organization. 2019. ISO/IEC 9421-210:2019, Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems. https://www.iso.org/standard/52075.html

[54] Mehdi Ousmer, Arthur Sluÿters, Nathan Magrofuoco, Paolo Roselli, and Jean Vanderdonckt. 2022. A Systematic Procedure for Comparing Template-Based Gesture Recognizers. In *Proceedings of the 24th International Conference on Human-Computer Interaction, HCI International 2022, June 26 - July 1, 2022 (Lecture Notes in Computer Science, Vol. 13519)*, Masaaki Kurosu, Sakae Yamamoto, Hirohiko Mori, Dylan D. Schmorrow, Cali M. Fidopiastis, Norbert A. Streitz, and Shin'ichi Konomi (Eds.). Springer, 160–179. https://doi.org/10.1007/978-3-031-17618-0_13

[55] Otto Parra, Sergio España, Jose Ignacio Panach, and Oscar Pastor. 2019. An empirical comparative evaluation of gestUI to include gesture-based interaction in user interfaces. *Science of Computer Programming* 172 (2019), 232–263. https://doi.org/10.1016/j.scico.2018.12.001

[56] Yosra Rekik, Radu-Daniel Vatavu, and Laurent Grisoni. 2014. Match-up & Conquer: A Two-Step Technique for Recognizing Unconstrained Bimanual and Multi-Finger Touch Input. In *Proc. of the ACM Int. Working Conference on Advanced Visual Interfaces* (Como, Italy) *(AVI '14)*. Association for Computing Machinery, New York, NY, USA, 201–208. https://doi.org/10.1145/2598153.2598167

[57] Lior Rokach. 2010. Ensemble-based classifiers. *Artif. Intell. Rev.* 33, 1-2 (2010), 1–39. https://doi.org/10.1007/S10462-009-9124-7

[58] Dean Rubine. 1991. Specifying Gestures by Example. In *Proc. of the 18th Annual Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH '91)*. ACM, New York, NY, USA, 329–337. https://doi.org/10.1145/122718.122753

[59] Luis Ricardo Ruiz, Fernando De la Rosa R., and José Tiberio Hernández. 2012. Platform integrating interactive applications with gesture-based interaction. In *2012 XXXVIII Conferencia Latinoamericana En Informatica* (Medellin, Colombia) *(CLEI '12)*. 1–9. https://doi.org/10.1109/CLEI.2012.6427129

[60] Priscila T. M. Saito, Rodrigo Y. M. Nakamura, Willian P. Amorim, João P. Papa, Pedro J. de Rezende, and Alexandre X. Falcão. 2015. Choosing the Most Effective Pattern Classification Model under Learning-Time Constraint. *PLOS ONE* 10, 6 (06 2015), 1–23. https://doi.org/10.1371/journal.pone.0129947

[61] Ugo Braga Sangiorgi, Vivian Genaro Motti, François Beuvens, and Jean Vanderdonckt. 2012. Assessing lag perception in electronic sketching. In *Proc. of ACM Int. Nordic Conf. on Human-Computer Interaction* (Copenhagen, Denmark) *(NordiCHI '12)*, Lone Malmborg and Thomas Pederson (Eds.). Association for Computing Machinery, New York, NY, USA, 153–161. https://doi.org/10.1145/2399016.2399040

[62] Benedikt Severin, Ole Werger, and Marc Hesenius. 2024. Not What I was Trained for – Out-of-Distribution-Tests for Interactive AIs. In *Engineering Interactive Computer Systems. EICS 2023 International Workshops and Doctoral Consortium: Swansea, UK, June 26-27, 2023, Selected Papers* (Swansea, United Kingdom). Springer-Verlag, Berlin, Heidelberg, 127–147. https://doi.org/10.1007/978-3-031-59235-5_12

[63] Alex Shaw, Jaime Ruiz, and Lisa Anthony. 2021. A Survey on Applying Automated Recognition of Touchscreen Stroke Gestures to Children's Input. *Interacting with Computers* 32, 5-6 (04 2021), 524–547. https://doi.org/10.1093/iwc/iwab009 arXiv:https://academic.oup.com/iwc/article-pdf/32/5-6/524/38856860/iwab009.pdf

[64] Junxiao Shen, John Dudley, George Mo, and Per Ola Kristensson. 2022. Gesture Spotter: A Rapid Prototyping Tool for Key Gesture Spotting in Virtual and Augmented Reality Applications. *IEEE Transactions on Visualization and Computer Graphics* 28, 11 (2022), 3618–3628. https://doi.org/10.1109/TVCG.2022.3203004

[65] Beat Signer, U. Kurmann, and Moira C. Norrie. 2007. iGesture: A General Gesture Recognition Framework. In *9th International Conference on Document Analysis and Recognition (ICDAR 2007), 23-26 September, Curitiba, Paraná, Brazil*. IEEE Computer Society, 954–958. https://doi.org/10.1109/ICDAR.2007.4377056

[66] Arthur Sluÿters, Mehdi Ousmer, Paolo Roselli, and Jean Vanderdonckt. 2022. QuantumLeap, a Framework for Engineering Gestural User Interfaces based on the Leap Motion Controller. *Proc. ACM Hum. Comput. Interact.* 6, EICS (2022), 161:1–161:47. https://doi.org/10.1145/3532211

[67] Arthur Sluÿters, Quentin Sellier, Jean Vanderdonckt, Vik Parthiban, and Pattie Maes. 2022. Consistent, Continuous, and Customizable Mid-Air Gesture Interaction for Browsing Multimedia Objects on Large Displays. *International Journal of Human–Computer Interaction* 0, 0 (2022), 1–32. https://doi.org/10.1080/10447318.2022.2078464 arXiv:https://doi.org/10.1080/10447318.2022.2078464

[68] Kênia Sousa, Hildeberto Mendonça, Jean Vanderdonckt, Els Rogier, and Joannes Vandermeulen. 2008. User Interface Derivation from Business Processes: A Model-Driven Approach for Organizational Engineering. In *Proceedings of the 2008 ACM Symposium on Applied Computing* (Fortaleza, Ceara, Brazil) *(SAC '08)*. Association for Computing Machinery, New York, NY, USA, 553–560. https://doi.org/10.1145/1363686.1363821

[69] M. Stone. 1974. Cross-Validatory Choice and Assessment of Statistical Predictions. *Journal of the Royal Statistical Society: Series B (Methodological)* 36, 2 (1974), 111–133. https://doi.org/10.1111/j.2517-6161.1974.tb00994.x

[70] Eugene M. Taranta, Mehran Maghoumi, Corey R. Pittman, and Joseph J. LaViola. 2016. A Rapid Prototyping Approach to Synthetic Data Generation for Improved 2D Gesture Recognition. In *Proc. of the 29th Annual Symposium on User Interface Software and Technology* (Tokyo, Japan) *(UIST '16)*. Association for Computing Machinery, New York, NY, USA, 873––885. https://doi.org/10.1145/2984511.2984525

[71] Eugene M. Taranta, II and Joseph J. LaViola, Jr. 2015. Penny Pincher: A Blazing Fast, Highly Accurate $-family Recognizer. In *Proc. of the 41st Graphics Interface Conference* (Halifax, Nova Scotia, Canada) *(GI '15)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 195–202. http://dl.acm.org/citation.cfm?id=2788890.2788925

[72] Eugene M. Taranta II, Amirreza Samiei, Mehran Maghoumi, Pooya Khaloo, Corey R. Pittman, and Joseph J. LaViola Jr. 2017. Jackknife: A Reliable Recognizer with Few Samples and Many Modalities. In *Proc. of the ACM Int. Conf. on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 5850–5861. https://doi.org/10.1145/3025453.3026002

[73] Andreas Theissler, Simon Vollert, Patrick Benz, Laurentius Antonius Meerhoff, and Marc Fernandes. 2020. ML-ModelExplorer: An Explorative Model-Agnostic Approach to Evaluate and Compare Multi-class Classifiers. In *Machine Learning and Knowledge Extraction - 4th IFIP TC 5, TC 12, WG 8.4, WG 8.9, WG 12.9 International Cross-Domain Conference, CD-MAKE 2020, Dublin, Ireland, August 25-28, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12279)*, Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar R. Weippl (Eds.). Springer, 281–300. https://doi.org/10.1007/978-3-030-57321-8_16

[74] Huawei Tu, Xiangshi Ren, and Shumin Zhai. 2012. A comparative evaluation of finger and pen stroke gestures. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 1287–1296. https://doi.org/10.1145/2207676.2208584

[75] Jean Vanderdonckt, Paolo Roselli, and Jorge Luis Pérez-Medina. 2018. !FTL, an Articulation-Invariant Stroke Gesture Recognizer with Controllable Position, Scale, and Rotation Invariances. In *Proc. of the 20th ACM Int. Conf. on Multimodal Interaction* (Boulder, CO, USA) *(ICMI '18)*. Association for Computing Machinery, New York, NY, USA, 125—-134. https://doi.org/10.1145/3242969.3243032

[76] Radu-Daniel Vatavu. 2011. The Effect of Sampling Rate on the Performance of Template-Based Gesture Recognizers. In *Proc. of the 13th Int. Conf. on Multimodal Interaction* (Alicante, Spain) *(ICMI '11)*. Association for Computing Machinery, New York, NY, USA, 271—-278. https://doi.org/10.1145/2070481.2070531

[77] Radu-Daniel Vatavu. 2012. 1F: One Accessory Feature Design for Gesture Recognizers. In *Proc. of the ACM Int. Conf. on Intelligent User Interfaces* (Lisbon, Portugal) *(IUI '12)*. Association for Computing Machinery, New York, NY, USA, 297–300. https://doi.org/10.1145/2166966.2167022

[78] Radu-Daniel Vatavu. 2017. Improving Gesture Recognition Accuracy on Touch Screens for Users with Low Vision. In *Proc. of the ACM Int. Conf. on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 4667–4679. https://doi.org/10.1145/3025453.3025941

[79] Radu-Daniel Vatavu. 2020. Gesture-Based Interaction. In *Handbook of Human-Computer Interaction*, Jean Vanderdonckt, Philippe Palanque, and Marco Winckler (Eds.). Springer International Publishing, Cham, 1–47. https://doi.org/10.1007/978-3-319-27648-9_20-1

[80] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. 2012. Gestures as Point Clouds: A $P Recognizer for User Interface Prototypes. In *Proc. of the 14th ACM Int. Confe. on Multimodal Interaction* (Santa Monica, California, USA) *(ICMI '12)*. Association for Computing Machinery, New York, NY, USA, 273–280. https://doi.org/10.1145/2388676.2388732

[81] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. 2013. Relative accuracy measures for stroke gestures. In *Proceedings of the 15th ACM on International Conference on Multimodal Interaction* (Sydney, Australia) *(ICMI '13)*. Association for Computing Machinery, New York, NY, USA, 279–286. https://doi.org/10.1145/2522848.2522875

[82] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. 2018. $Q: A Super-quick, Articulation-invariant Stroke-gesture Recognizer for Low-resource Devices. In *Proc. of the 20th Int. Conf. on Human-Computer Interaction with Mobile Devices and Services* (Barcelona, Spain) *(MobileHCI '18)*. Association for Computing Machinery, New York, NY, USA, Article 23, 12 pages. https://doi.org/10.1145/3229434.3229465

[83] Radu-Daniel Vatavu, Daniel Vogel, Géry Casiez, and Laurent Grisoni. 2011. Estimating the Perceived Difficulty of Pen Gestures. In *Proc. of the 13th IFIP TC 13 Int. Conf. on Human-Computer Interaction, Part II* (Lisbon, Portugal) *(INTERACT '11)*. Springer, Berlin, Heidelberg, 89—-106. https://doi.org/10.1007/978-3-642-23771-3_9

[84] Santiago Villarreal-Narvaez, Arthur Sluÿters, Jean Vanderdonckt, and Radu-Daniel Vatavu. 2024. Brave New GES World: A Systematic Literature Review of Gestures and Referents in Gesture Elicitation Studies. *ACM Comput. Surv.* 56, 5, Article 128 (Jan. 2024), 55 pages. https://doi.org/10.1145/3636458

[85] Thomas Wetzlmaier and Mario Winterer. 2015. Test automation for multi-touch user interfaces of industrial applications. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 1–3. https://doi.org/10.1109/ICSTW.2015.7107468

[86] Don Willems, Ralph Niels, Marcel van Gerven, and Louis Vuurpijl. 2009. Iconic and multi-stroke gesture recognition. *Pattern Recognition* 42, 12 (2009), 3303–3312. https://doi.org/10.1016/j.patcog.2009.01.030 New Frontiers in Handwriting Recognition.

[87] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. 2007. Gestures Without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *Proc. of the 20th Annual ACM Symposium on User Interface Software and Technology* (Newport, Rhode Island, USA) *(UIST '07)*. Association for Computing Machinery, New York, NY, USA, 159–168. https://doi.org/10.1145/1294211.1294238

[88] Carl D. Worth. 2003. xstroke: Full-screen Gesture Recognition for X. In *Proceedings of the FREENIX Track: 2003 USENIX Annual Technical Conference, June 9-14, 2003, San Antonio, Texas, USA*. 187–196. http://www.usenix.org/events/usenix03/tech/freenix03/worth.html

[89] Haijun Xia, Michael Glueck, Michelle Annett, Michael Wang, and Daniel Wigdor. 2022. Iteratively Designing Gesture Vocabularies: A Survey and Analysis of Best Practices in the HCI Literature. *ACM Trans. Comput. Hum. Interact.* 29, 4 (2022), 37:1–37:54. https://doi.org/10.1145/3503537

[90] Momona Yamagami, Claire L. Mitchell, Alexandra A Portnova-Fahreeva, Junhan Kong, Jennifer Mankoff, and Jacob O. Wobbrock. 2024. Customized Mid-Air Gestures for Accessibility: A $B Recognizer for Multi-Dimensional Biosignal Gestures. *CoRR* abs/2409.08402 (2024). https://doi.org/10.48550/ARXIV.2409.08402 arXiv:2409.08402

[91] Yulong Yang, Gradeigh D. Clark, Janne Lindqvist, and Antti Oulasvirta. 2016. Free-Form Gesture Authentication in the Wild. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 3722–3735. https://doi.org/10.1145/2858036.2858270

[92] Ying Yang, Geoff Webb, Kevin Korb, and Kai Ming Ting. 2007. Classifying under computational resource constraints: anytime classification using probabilistic estimators. *Machine Learning* 69 (10 2007), 35–53. https://doi.org/10.1007/s10994-007-5020-z

[93] Yina Ye and Petteri Nurmi. 2015. Gestimator: Shape and Stroke Similarity Based Gesture Recognition. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction* (Seattle, Washington, USA) *(ICMI '15)*. Association for Computing Machinery, New York, NY, USA, 219–226. https://doi.org/10.1145/2818346.2820734

[94] Shumin Zhai, Per Ola Kristensson, Caroline Appert, Tue Haste Anderson, and Xiang Cao. 2012. Foundational Issues in Touch-Surface Stroke Gesture Design — An Integrative Review. *Foundations and Trends in Human–Computer Interaction* 5, 2 (2012), 97–205. https://doi.org/10.1561/1100000012

[95] Yougen Zhang, Wei Deng, Hanchen Song, and Lingda Wu. 2013. A Fast Pen Gesture Matching Method Based on Nonlinear Embedding. In *Advances in Image and Graphics Technologies*, Tieniu Tan, Qiuqi Ruan, Xilin Chen, Huimin Ma, and Liang Wang (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 223–231. https://doi.org/10.1007/978-3-642-37149-3_27

A The $C recognizer based on Clifford Algebra

The convergence of $C is demonstrated in Luzzi and Roselli [45] with questions such as: if a candidate gesture is translated, does the $C score change?; if a candidate gesture is uniformly scaled, does the $C score change?; if a candidate gesture is rotated, does the $C score change?; if one increases the number of sampled points of the two gestures, does the $C score have a limit value?; if such a limit value exists, can we find a closed formula to express it? $C is invariant to translation, scaling, and rotation; moreover, under certain hypotheses, increasing the number of sampled points improves the $C score, which corresponds to a well-defined number explicitly expressed as a Riemann integral [12, 45].

Fig. 5 graphically shows how an input template is normalized in $C and Fig. 6 illustrates an example of an on-line recognition of a candidate gesture, producing a score of S=4.4.

Fig. 5. Normalization of an input template into $C representation.

We hereby define the formula used by the recognizer to compute the distance between a candidate gesture and a template:

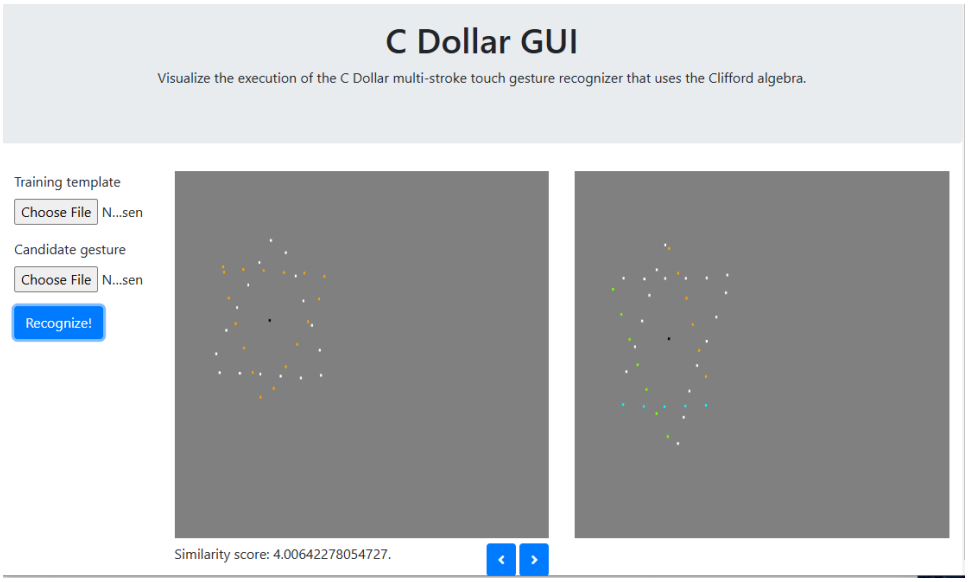
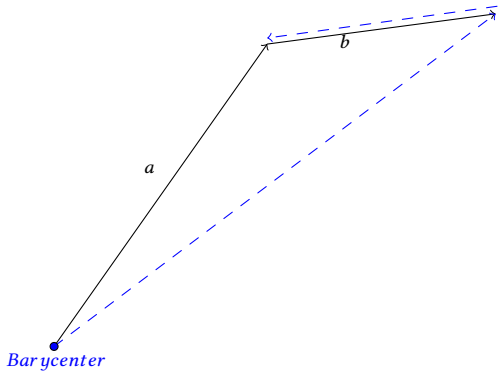


Fig. 6. On-line recognition of a candidate gesture by the $C recognizer.



$$Sym_{\$C}LSD(a,b,u,v) = \min\left\{LSD(a,b,u,v)\ ,\ LSD(a+b,-b,u,v)\right\}$$

$$LSD^2(a,b,u,v) = \frac{a^2v^2 + b^2u^2 - 2\left[(a\cdot b)(u\cdot v) - (a\cdot v)(b\cdot u) + (a\cdot u)(b\cdot v)\right]}{b^2v^2}$$

$$LSD^2(a+b,-b,u,v) = \frac{(a+b)^2v^2 + (-b)^2u^2 - 2\Big[\big((a+b)\cdot(-b)\big)(u\cdot v) - \big((a+b)\cdot v\big)\big((-b)\cdot u\big) + \big((a+b)\cdot u\big)\big((-b)\cdot v\big)\Big]}{b^2v^2} =$$

$$= \frac{(a^2+b^2+2a\cdot b)v^2 + b^2u^2 - 2\Big[-(a\cdot b)(u\cdot v) - b^2(u\cdot v) + (a\cdot v)(b\cdot u) + \cancel{(b\cdot v)(b\cdot u)} - (a\cdot u)(b\cdot v) - \cancel{(b\cdot u}$$

$$= \frac{a^2v^2 + b^2u^2 - 2\Big[-(a\cdot b)(u\cdot v) + (a\cdot v)(b\cdot u) - (a\cdot u)(b\cdot v)\Big]}{b^2v^2} + \frac{(b^2+2a\cdot b)v^2 - 2\Big[-b^2(u\cdot v)\Big]}{b^2v^2} =$$

$$= \frac{a^2v^2 + b^2u^2 + 2\Big[(a\cdot b)(u\cdot v) - (a\cdot v)(b\cdot u) + (a\cdot u)(b\cdot v)\Big]}{b^2v^2} + 1 + 2\left(\frac{a\cdot b}{b^2} + \frac{u\cdot v}{v^2}\right)$$

If

$$C = (a\cdot b)(u\cdot v) - (a\cdot v)(b\cdot u) + (a\cdot u)(b\cdot v)$$

then

$$LSD^2(a,b,u,v) = \frac{a^2v^2 + b^2u^2 - 2C}{b^2v^2}$$

$$LSD^2(a+b,-b,u,v) = \frac{a^2v^2 + b^2u^2 + 2C}{b^2v^2} + 1 + 2\left(\frac{a\cdot b}{b^2} + \frac{u\cdot v}{v^2}\right)$$

Thus,

$$LSD(a,b,u,v) < LSD(a+b,-b,u,v)$$

if and only if

$$\frac{a^2v^2 + b^2u^2 - 2C}{b^2v^2} < \frac{a^2v^2 + b^2u^2 + 2C}{b^2v^2} + 1 + 2\left(\frac{a\cdot b}{b^2} + \frac{u\cdot v}{v^2}\right),$$

that is, $\boxed{4C + b^2v^2 + 2\Big[(a\cdot b)v^2 + (u\cdot v)b^2\Big] > 0}$.

Thus, in order to establish if

$$Sym_{\$C}LSD(a,b,u,v) = LSD(a,b,u,v) \;\text{ or }\; Sym_{\$C}LSD(a,b,u,v) = LSD(a+b,-b,u,v)\,,$$

it suffices to evaluate the sign of

$$4C + b^2v^2 + 2\Big[(a\cdot b)v^2 + (u\cdot v)b^2\Big]\,.$$

Received 25 October 2024; revised 25 December 2024; revised 14 February 2025; revised 4 March 2025; revised 27 April 2025