Validation of Composite Systems by Discrepancy Propagation (Supplementary Material)

David Reeb[1] **Kanil Patel**[1] **Karim Barsim**[1] **Martin Schiegg**[1] **Sebastian Gerwinn**[1]

[1]Bosch Center for Artificial Intelligence, Robert Bosch GmbH, 71272 Renningen, Germany

A SEMIDEFINITE RELAXATION OF THE BOUND OPTIMIZATION IN EQ. (4)

Let the discrepancy measures in Eq. (4) in Sec. 3.2 be given by MMD (maximum mean discrepancy) with kernels $k^{c\to c''}$ and $k^{c'\to c}$, respectively. When representing the distributions via samples, we have (see [Gretton et al., 2012]):

$$D(p_\alpha|_{c\to c''}, q|_{c\to c''})^2 = \alpha^\top \mathbf{K}^{VV}_{c\to c''}\alpha - 2\alpha^\top \mathbf{K}^{VM}_{c\to c''}\frac{\mathbf{e}_{n_M}}{n_M} + \frac{\mathbf{e}_{n_M}^\top}{n_M}\mathbf{K}^{MM}_{c\to c''}\frac{\mathbf{e}_{n_M}}{n_M} \tag{1}$$

with kernel matrices (with indices $v, v' = 1, \ldots, V^c$, $n, n' = 1, \ldots, n_M$)

$$\begin{aligned}
\left(\mathbf{K}^{VV}_{c\to c''}\right)_{v,v'} &:= k^{c\to c''}\left(y^c_v, y^c_{v'}\right), \\
\left(\mathbf{K}^{VM}_{c\to c''}\right)_{v,n'} &:= k^{c\to c''}\left(y^c_v, y^{M^c}_{n'}\right), \\
\left(\mathbf{K}^{MM}_{c\to c''}\right)_{n,n'} &:= k^{c\to c''}\left(y^{M^c}_n, y^{M^c}_{n'}\right),
\end{aligned}$$

and where $\mathbf{e}_d := (1, \ldots, 1)^\top \in \mathbb{R}^d$ denotes the d-dimensional all-1's vector, so that $\frac{\mathbf{e}_d}{d}$ is the uniform probability vector on d elements. Similarly, at the input of component c we have

$$D(p_\alpha|_{c'\to c}, q|_{c'\to c})^2 = \alpha^\top \mathbf{K}^{VV}_{c'\to c}\alpha - 2\alpha^\top \mathbf{K}^{VM}_{c'\to c}\frac{\mathbf{e}_{n_M}}{n_M} + \frac{\mathbf{e}_{n_M}^\top}{n_M}\mathbf{K}^{MM}_{c'\to c}\frac{\mathbf{e}_{n_M}}{n_M}, \tag{2}$$

where by a slight abuse of notation we define the input kernel matrices as

$$\begin{aligned}
\left(\mathbf{K}^{VV}_{c'\to c}\right)_{v,v'} &:= k^{c'\to c}\left(x^c_v, x^c_{v'}\right), \\
\left(\mathbf{K}^{VM}_{c'\to c}\right)_{v,n'} &:= k^{c'\to c}\left(x^c_v, x^{M^c}_{n'}\right), \\
\left(\mathbf{K}^{MM}_{c'\to c}\right)_{n,n'} &:= k^{c'\to c}\left(x^{M^c}_n, x^{M^c}_{n'}\right).
\end{aligned}$$

Taken together, we can write (4) in Sec. 3.2 – or rather its square – as the following quadratic optimization problem:

$$\begin{aligned}
&(B^{c\to c''})^2 && (3)\\
&= \sup_{\{p: D(p_\alpha|_{c'\to c}, q|_{c'\to c})^2 \le (B^{c'\to c})^2 \quad \forall c' < c\}} D(p_\alpha|_{c\to c''}, q|_{c\to c''})^2 && (4)\\
&= \text{maximize}_\alpha\ \alpha^\top \mathbf{K}^{VV}_{c\to c''}\alpha - 2\alpha^\top \mathbf{K}^{VM}_{c\to c''}\frac{\mathbf{e}_{n_M}}{n_M} + \frac{\mathbf{e}_{n_M}^\top}{n_M}\mathbf{K}^{MM}_{c\to c''}\frac{\mathbf{e}_{n_M}}{n_M} && (5)\\
&\qquad \text{subject to}\ \ \alpha^\top \mathbf{K}^{VV}_{c'\to c}\alpha - 2\alpha^\top \mathbf{K}^{VM}_{c'\to c}\frac{\mathbf{e}_{n_M}}{n_M} + \frac{\mathbf{e}_{n_M}^\top}{n_M}\mathbf{K}^{MM}_{c'\to c}\frac{\mathbf{e}_{n_M}}{n_M} \le (B^{c'\to c})^2 \quad \forall c' < c, && (6)\\
&\qquad\qquad \mathbf{e}_{V^c}^\top \alpha = 1, && (7)\\
&\qquad\qquad \alpha \ge 0, && (8)
\end{aligned}$$

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

where the vector constraint $\alpha \geq 0$ is understood entry-wise.

Unfortunately, the optimization problem (5–8) is *not* a convex optimization problem because the objective is to maximize a convex function. While heuristic solvers are available, such as the package `qcqp` [Park and Boyd, 2017], we avoid those heuristic methods, as they do not guarantee a valid upper bound and can be inefficient in computation.

Instead we follow approaches more tailored to the problem, and relax the original problem (5–8) to obtain an efficiently solvable semidefinite program (SDP), a type of convex optimization problem. We follow the "tightened semidefinite relaxations" from Park and Boyd [2017][Secs. 3.3, 3.4].

For this, we introduce the (symmetric) matrix variable $A := \alpha\alpha^\top$ and rewrite the quadratic terms in (5) and (6) via the matrix traces $\alpha^\top \mathbf{K}^{VV}_{c\to c''}\alpha = \mathrm{Tr}[\mathbf{K}^{VV}_{c\to c''}A]$ and $\alpha^\top \mathbf{K}^{VV}_{c'\to c}\alpha = \mathrm{Tr}[\mathbf{K}^{VV}_{c'\to c}A]$, so that they are now *linear* in the variable A. Due to (8), $A = \alpha\alpha^\top$ is entry-wise nonnegative, which we write as $A \geqslant 0$; furthermore, it holds that $\mathbf{e}_{V_c}^\top A \mathbf{e}_{V_c} = 1$ due to (7). Also due to (7), we can even recover α from $A = \alpha\alpha^\top$ via $\alpha = A\mathbf{e}_{V_c}$. Therefore, we will omit our original variable α in favor of the matrix variable A and simply *define* the expression $\alpha := A\mathbf{e}_{V_c}$. Rewriting (5–8) in terms of A and additionally adding the constraints $A = \alpha\alpha^\top$, $\mathbf{e}_{V_c}^\top A\mathbf{e}_{V_c} = 1$ and $A \succcurlyeq 0$ gives the same optimum as (5–8). The resulting (rewritten) problem is convex except for the equality constraint $A = \alpha\alpha^\top$. We finally relax this constraint to the matrix inequality $A \geq \alpha\alpha^\top$ (where the inequality is understood w.r.t. the positive semidefinite order), which *is* convex. Writing this matrix inequality in the manifestly convex way (12) as a semidefinite constraint [Park and Boyd, 2017], we therefore obtain:

Lemma 1 (Tightened SDP relaxation of (5–8))**.** *Define the tightened SDP relaxation of (5–8) as the following semidefinite optimization problem (SDP) with symmetric matrix variable $A = A^\top \in \mathbb{R}^{V^c\times V^c}$ and the abbreviation $\alpha := A\mathbf{e}_{V_c}$:*

$$
\begin{aligned}
(B^{c\to c''}_{\textit{SDR-tightened}})^2 \quad := & && (9)\\
\textit{maximize}_A \quad & \mathrm{Tr}[\mathbf{K}^{VV}_{c\to c''}A] - 2\alpha^\top \mathbf{K}^{VM}_{c\to c''}\frac{\mathbf{e}_{n_M}}{n_M} + \frac{\mathbf{e}_{n_M}^\top}{n_M}\mathbf{K}^{MM}_{c\to c''}\frac{\mathbf{e}_{n_M}}{n_M} && (10)\\
\textit{subject to} \quad & \mathrm{Tr}[\mathbf{K}^{VV}_{c'\to c}A] - 2\alpha^\top \mathbf{K}^{V}_{c'\to c}\frac{\mathbf{e}_{n_M}}{n_M} + \frac{\mathbf{e}_{n_M}^\top}{n_M}\mathbf{K}^{MM}_{c'\to c}\frac{\mathbf{e}_{n_M}}{n_M} \leq (B^{c'\to c})^2 \quad \forall c' < c, && (11)\\
& \begin{pmatrix} A & \alpha \\ \alpha^\top & 1 \end{pmatrix} \geq 0 \ \textit{(i.e. the left-hand-side is a positive semidefinite matrix)}, && (12)\\
& \mathbf{e}_{V^c}^\top A \mathbf{e}_{V^c} = 1, && (13)\\
& A \succcurlyeq 0 \ \textit{(entry-wise; above the diagonal suffices due to constraint (12) and } A = A^\top\textit{)}. && (14)
\end{aligned}
$$

Then its optimal value satisfies $(B^{c\to c''}_{\textit{SDR-tightened}})^2 \geq (B^{c\to c''})^2$, i.e. $(B^{c\to c''}_{\textit{SDR-tightened}})^2$ is an upper bound on the optimum of the non-convex problem (5–8), which itself is (the square of) an upper bound on the (unknown) MMD discrepancy $D(p|_{c\to c''}, q|_{c\to c''})$ (cf. Eq. (4) in Sec. 3.2).

To solve the relaxed semidefinite programs from Lemma 1, we use the library `CVXPY` [Diamond and Boyd, 2016]. Note that due to the relaxation, the value $(B^{c\to c''}_{\text{SDR-tightened}})^2$ of the relaxed optimization from Lemma 1 is in general different (larger, i.e. worse) than the $(B^{c\to c''})^2$ from the original optimization (5); this can happen if the found optimum $\widehat{A}$ of the relaxed problem in Lemma 1 cannot be expressed as $\widehat{A} = \widehat{\alpha}\widehat{\alpha}^\top$ (i.e. the optimal $\widehat{A}$ is not of rank 1). To detect such a *relaxation gap*, one can plug the found vector $\widehat{\alpha} := \widehat{A}\mathbf{e}_{V_c}$ into (5) and compare its value $\mathrm{Opt}_{\text{orig}}(\widehat{\alpha})$ to the optimal value $\mathrm{Opt}_{\text{relax}}(\widehat{A}) = (B^{c\to c''}_{\text{SDR-tightened}})^2$ of (10). Then, by Lemma 1 and the maximization (5), it holds that the true optimum $(B^{c\to c''})^2$ of the nonconvex problem (5) is sandwiched between two convexly computable quantities:

$$
\mathrm{Opt}_{\text{orig}}(\widehat{\alpha}) \ \leq \ (B^{c\to c''})^2 \ \leq \ \mathrm{Opt}_{\text{relax}}(\widehat{A}) = (B^{c\to c''}_{\text{SDR-tightened}})^2. \tag{15}
$$

When both values agree, $\mathrm{Opt}_{\text{orig}}(\widehat{\alpha}) = \mathrm{Opt}_{\text{relax}}(\widehat{A})$, then the relaxation was *tight*, i.e. we have certified that $(B^{c\to c''})^2 = (B^{c\to c''}_{\text{SDR-tightened}})^2 = \mathrm{Opt}_{\text{relax}}(\widehat{A})$ are equal and our relaxation has found the true optimum. More generally, we can guarantee a *relaxation gap* $\Delta = (B^{c\to c''}_{\text{SDR-tightened}})^2 - (B^{c\to c''})^2$ of at most $\widehat{\Delta} = \mathrm{Opt}_{\text{relax}}(\widehat{A}) - \mathrm{Opt}_{\text{orig}}(\widehat{\alpha})$, and an approximation ratio $\gamma = (B^{c\to c''})^2/(B^{c\to c''}_{\text{SDR-tightened}})^2 \in [0,1]$ of at least $\widehat{\gamma} = \mathrm{Opt}_{\text{orig}}(\widehat{\alpha})/\mathrm{Opt}_{\text{relax}}(\widehat{A}) \in [0,1]$.

Empirically we find in this way that the relaxation in Lemma 1 is basically tight in most problem instances, i.e. it returns (almost) the correct optimum of (5–8). See App. E.4 for an empirical evaluation.

With this relaxation, the number of optimization variables increases from $\dim(\alpha) = V^c$ in (5–8) to $\dim(A = A^\top) = V^c(V^c+1)/2 \sim (V^c)^2/2$ in (10–14), i.e. it grows quadratically with the number of validation points V^c for component c; the number of optimization constraints also increases like $\sim C + (V^c)^2$. This results in a computational limitation which restricts the number of validation inputs to roughly $V^c \lesssim 10^3$ with standard convex solvers on standard computing hardware. If one would like to apply our general validation method with more validation points V^c, one would have to to find another way to efficiently upper-bound the optimization problem (5–8), instead of our tightened SDP relaxation (Lemma 1).

To run the whole validation method in Algorithm 1, the number of bound optimizations (5–8) or (10–14) to perform equals the number of connections $c \to c''$ between components of the system (where $1 \leq c < c'' \leq C+1$, see Sec. 3.1), i.e. the number of such bound computation lies between C (for the linear chain) and $C(C+1)/2$ (for the "fully connected" system). See App. E.5 for actual runtimes of our method.

B DETAILS ON THE FAILURE PROBABILITY OPTIMIZATION IN EQ. (5)

Similar to (5–8), we can formulate a sample-based optimization for the optimization problem in Eq. (5) in Sec. 3.2 in the case of using MMD as the discrepancy measure, to compute an upper bound F_{max} on the system failure probability p_{fail}. More precisely, assuming that this MMD measure on the TPI output y^C has kernel $k^y \equiv k^{C\to C+1}$, we can write this problem (optionally with the monotonicity and Lipschitz conditions mentioned below Eq. (5) in Sec. 3.2 with the optimization variable $\alpha \in \mathbb{R}^V$:

$$F_{\text{max}} = \text{maximize}_\alpha \sum_{v: g_v > \tau} \alpha_v \tag{16}$$

$$\text{subject to } \alpha^\top \mathbf{K}_y^{VV} \alpha - 2\alpha^\top \mathbf{K}_y^{VM} \frac{\mathbf{e}_{n_M}}{n_M} + \frac{\mathbf{e}_{n_M}^\top}{n_M} \mathbf{K}_y^{MM} \frac{\mathbf{e}_{n_M}}{n_M} \leq (B^y)^2, \tag{17}$$

$$\alpha \geq 0, \qquad \mathbf{e}_V^\top \alpha = 1, \tag{18}$$

$$\alpha_v \leq \alpha_{v-1} \quad \forall v \text{ with } g_v \geq \tau' \quad \text{(monotonicity; we take } \tau' := \tau), \tag{19}$$

$$|\alpha_{v+1} - \alpha_v| \leq \Lambda_{\text{max}} |g_{v+1} - g_v| \quad \forall v \quad \text{(Lipschitz condition)}, \tag{20}$$

where we defined kernel matrices on the TPI grid-points g_v and simulation outputs y_n^M:

$$(\mathbf{K}_y^{VV})_{v,v'} = k^y\left(g_v, g_{v'}\right),$$
$$(\mathbf{K}_y^{VM})_{v,n'} = k^y\left(g_v, y_{n'}^M\right),$$
$$(\mathbf{K}_y^{MM})_{n,n'} = k^y\left(y_n^M, y_{n'}^M\right).$$

The optimization problem (16–20) has a linear objective and linear constraints except for the quadratic MMD constraint (17). It is thus a convex optimization problem that can be solved efficiently and exactly, without relaxations (unlike required for the bound optimization (4) in Sec. 3.2, see App. A).

The above formulation depends on a set of grid-points g_v and a Lipschitz constant Λ_{max}. Ideally, the Λ_{max} should be a tight upper bound on the Lipschitz constant of the system's TPI output density p_y. As we do not know this density p_y, we use a heuristic estimator of Λ_{max} computed from histograms of the simulation output distribution q_y as a proxy (see experimental details in App. D). In order for the solution of (16–20) to be close to its true value (5) in Sec. 3.2, the grid-points need to be sufficiently dense in order to reveal differences as measured by the MMD constraint (17) (see also App. C). As the MMD measure $D(p_y, q_y)$ with kernel k^y can also be interpreted as the L2-distance between the corresponding kernel density estimators of the samples from p_y and q_y [Gretton et al., 2012], we choose the grid-spacing relative to the lengthscale ℓ of the kernel k^y; more precisely, in our experiments we require that $g_{v+1} - g_v \leq \frac{\ell}{5}$. Additionally, the grid-points should cover the range where the support of both p_y and q_y lies (although p_y is not known). For our experiments, we chose the grid range $[g_1, g_V]$ such that it contains all data-points from q_y, as well as a significantly large region around the threshold τ (see App. D). Even though the system's TPI output distribution p_y is not known, in many applications the plausible (or even the potential) range of TPI values y^C will typically be known from domain knowledge; in this case, the grid endpoints should be chosen to cover this range.

Note, that the optimization for the failure probability in Eq. (5) in Sec. 3.2 or in Eqs. (16–20) is also possible for higher-dimensional TPI quantities y^C. In this case, a specification $y^C \in \mathcal{TPI}_{\text{fail}}$ of the critical region is required (replacing the

specification $y^C > \tau$ in (16)). Even if this specification is non-linear, the optimization objective will remain linear and the constraints quadratic, again by choosing a grid on the TPI space as in the one-dimensional case.

By construction, the final solution F_{max} (computed via (4),(5 in Sec. 3.2), or more concretely via (10–14),(16–20)) is an upper bound on the system's true failure probability $F_{\text{max}} \geq p_{\text{fail}} = \int \mathbb{1}_{y>\tau} dp_y(y) = \int \mathbb{1}_{S(x)>\tau} dS(x) dp_x(x)$ (see Prop. 1 in Sec. 3.2 and also App. C); as a result it can be used for virtual system validation. This bound F_{max} remains valid for any discrepancy measure, choice of kernels or lengthscales. For example, when choosing MMD with a very large kernel lengthscale as discrepancy measure, its discriminative power is minimal, resulting in very small discrepancy values and hence small corresponding bounds $B^{c\to c''}$. However, for such a measure, it is also difficult to distinguish p_α from the given $q|_{c\to c''}$ in a later bound propagation step, counteracting the small obtained $B^{c\to c''}$ and potentially resulting in a larger final F_{max}. We, therefore, use the final F_{max} as the minimization objective in a Bayesian Optimization scheme [Fröhlich et al., 2020] to select the kernel parameters, see also App. D. Another option to select good kernel parameters would be gradient-based minimization of F_{max} w.r.t. to the kernel parameters, which is possible in our framework as the convex programs from Apps. A and B can be (automatically) differentiated [Agrawal et al., 2019].

C PROOF AND EXTENSIONS OF PROP. 1 (SEC. 3.2)

Proof of Prop. 1. Under the assumptions *(i)* and *(ii)* that the set of validation inputs $\{x_v^c\}_v$ contains *all* actually occurring input points into S^c and that $p_\alpha = \sum_v \alpha_v \delta_{x_v^c} S^c(x_v^c)$ is built with the *correct* outputs $S^c(x_v^c)$, this set of distributions p_α over which we optimize in (4) (see Sec. 3.2) contains the joint real-world distribution of in- and outputs of S^c (i.e. the marginal of the real-world distribution p capturing the joint in- and outputs of S^c). Thus, if the input bound values $B^{c'\to c}$ were true upper bounds on the actual $D(p|_{c'\to c}, q|_{c'\to c})$, then $B^{c\to c''}$ from (4) in Sec. 3.2 is also a true upper bound on the actual $D(p|_{c\to c''}, q|_{c\to c''})$. By induction on $c = 1, 2, \ldots, C$ we can thus conclude that $B^y \equiv B^{C\to C+1}$ is a true upper bound on the real-world discrepancy $D(p_y, q_y) \equiv D(p|_{C\to C+1}, q|_{C\to C+1})$ if only the initial bound values $B^{0\to c}$ were true upper bounds on the actual initial discrepancies $D(p|_{0\to c}, q|_{0\to c})$; we assume this last statement about the initial bound values $B^{0\to c}$ to be true since they are supposed to be given in that way (alternatively, the same statement can be concluded with high confidence $\geq 1-\delta$ if the $B^{0\to c}$ were computed via samples from p_x [Gretton et al., 2012]; see end of Sec. 3.1). Finally, by the same reasoning, under the assumption *(iii)* that the set $\{g_v\}_v$ of grid-points contains *all* occurring real-world TPI values, the optimization (5) from Sec. 3.2 translates the true upper bound B^y into a true upper bound F_{max} on the real-world failure probability p_{fail}. □

Note that further upper-bounding the optimizations (4),(5) from Sec. 3.2 by relaxations as in App. A leads to valid upper bounds F_{max} as well, by the same reasoning as in the above proof.

The assumptions *(i)* and *(ii)* of Prop. 1 are so strong that one basically knows all system maps S^c *explicitly*, at least on all those inputs points that occur in the real world. If one would, in addition, know the real-world input distribution p_x (e.g. in a sample-based way), one could (in theory) simulate the system map S on all those samples and compute (or at least estimate) the real-world TPI distribution p_y by Monte-Carlo sampling; thus determine the desired p_{fail} arbitrarily well. However, we do *not* need the input distribution p_x to be known explicitly for our proposed method to be applicable; and we apply our method even when the strong knowledge about the S^c implied by *(i)* and *(ii)* is *not* available.

Remark 1 (Upper bounds in the limit)**.** Beyond the strong assumptions of Prop. 1, the upper bounds obtained by our method (4),(5) (see Sec. 3.2) can be proven to be valid under weaker, more realistic assumptions. This appears possible, for example, in the following scenario, as the numbers $V^c \equiv V$ of available validation data points grow: *(a)* The set of validation inputs $\{x_v^c\}_{v=1}^V$ covers the input space $\mathcal{S}_{in}^c \subset \mathbb{R}^{d_{in}^c}$ of S^c increasingly densely as $V \to \infty$, e.g. in the sense that $\max_{x\in\mathcal{S}_{in}^c} \min_{v\in\{1,\ldots,V\}} \|x - x_v^c\| \longrightarrow 0$ as $V \to \infty$; with an analogous condition for the set of grid-points $\{g_v\}_{v=1}^V$ used in (5) from Sec. 3.2. *(b)* As discrepancy measures $D^{c'\to c}$ we use MMD distances w.r.t. continuous and bounded kernel functions $k^{c'\to c}$. This is satisfied by all kernels used here, such as the squared-exponential and IMQ kernels, even when applied after a data embedding [Gretton et al., 2012]. *(c)* The real-world subsystem maps $S^c : x^c \mapsto S^c(x^c)$ (which are not known explicitly, and whose output is a probability distribution in general) are continuous w.r.t. the discrepancy measures $D^{c\to c''}$ at their output. This condition would be implied by continuity w.r.t. the Wasserstein or the total variation distances and might in some cases be argued from physical considerations. For deterministic components S^c, this requirement simply means that the deterministic mapping is continuous. *(d)* For each validation input x_v^c we have measured a sufficient number W of i.i.d. samples $y_{v,w}^c \sim S^c(x_v^c)$ $(w = 1, \ldots, W)$ from the *true but unknown* output distribution $S^c(x_v^c)$ and we use the sample-based $p_\alpha := \sum_{v=1}^V \sum_{w=1}^W \frac{\alpha_v}{W} \delta_{x_v^c} \delta_{y_{v,w}^c}$ in the optimization (4) from Sec. 3.2. We need $W = W(V)$ to be large enough that the empirical estimate $(1/W) \sum_{w=1}^W \delta_{y_{v,w}^c}$ is sufficiently close to $S^c(x_v^c)$ in the kernel mean embedding

[Muandet et al., 2017], instead of the exact equality required by Prop. 1. *(e)* The initial $B^{0\to c}$ must either be true upper bounds on the actual $D(p|_{0\to c}, q|_{0\to c})$, or must be computed empirically from an increasing number of i.i.d. samples $\{x_v\}_{v=1}^{V}$ coming from the real-world distribution p_x [Gretton et al., 2012] (see end of Sec. 3.1).

Even under such more realistic circumstances, one may still obtain a provably valid upper bound F_{max} on the real-world failure probability p_{fail} via our method in the limit $V \to \infty$ of sufficiently many validation points, at least almost surely over the sampling of $y^c_{v,w}$ (and x_v in the case where $B^{0\to c}$ are estimated from those samples) and up to any additive $\varepsilon > 0$ chosen beforehand. This can be shown by following the steps of the proof of Prop. 1, replacing each valid upper bound or exact equality by an approximation or limiting argument, using the above assumptions. Even more, when e.g. the kernels $k^{c'\to c}$ as well as Lipschitz constants of the maps S^c are known, then more effective statements can be obtained, in the sense that V can then be related to ε and to the confidence in the final F_{max} being a valid bound.

However, even such more realistic convergence statements would still not be practical in all cases, since e.g. the assumption *(a)* generally requires a number of validation inputs $V \gtrsim (1/\delta)^{d^c_{in}}$ *exponential* in the dimensions of the input spaces of the S^c (where $\delta = \max_{x\in\mathcal{S}^c_{in}} \min_{v\in\{1,\dots,V\}} \|x - x^c_v\|$ denotes the desired set approximation accuracy). We, therefore, take a pragmatic viewpoint, in that we apply our method (4),(5) (Sec. 3.2; see also Algorithm 1) even in those cases where we only have a limited amount of validation data available. We evalute this empirically in Sec. 4.2.

Remark 2 (Arbitrary simulation)**.** For the justification of our method – either heuristically or rigorously as above – it is *not* necessary that the "simulation distribution" q be in any sense close to the true system behavior or that the simulations M^c need to be faithful approximations of the actual system components S^c. Rather, it suffices that each of the distributions $q|_{\tilde{c}\to\hat{c}}$ (i.e. one distribution for each pair $(\tilde{c}, \hat{c})$ with $0 \le \tilde{c} < \hat{c} \le C+1$) has the same value in each of the optimizations (4) and (5) (Sec. 3.2); each of these distributions $q|_{\tilde{c}\to\hat{c}}$ simply acts as an "anchor" with respect to which the (unknown) real world distribution $p|_{\tilde{c}\to\hat{c}}$ is assessed – and these anchors need to remain fixed. This means that e.g. the full joint distribution q could have been generated by starting from an arbitrarily chosen q_x and arbitrarily "bad" models M^c, and that the resulting F_{max} should remain an upper bound on p_{fail} regardless. In practice we nevertheless desire the simulations M^c to be close to S^c as we intuitively believe that such closeness should lead to stronger statements about the system via the simulations, i.e. that the upper bound F_{max} be as good (i.e. small or tight) as possible. We investigate this dependency in the experiments (Sec. 4) via the comparison of Perfect Model vs. Misfit Model.

D DETAILS ON EXPERIMENTS (SEC. 4)

This section will provide all experiment details to reproduce the result shown in Sec. 4.2. It will contain the description of the reliability benchmark problems, the (input) data generation process, the per-channel/signal kernels and their length scales, the construction of the bias input (i.e. Biased Input), the misfit model (i.e. Misfit Model) and hyper-parameters of the failure probability computation. Most settings are kept fix across all benchmark problems (described in Sec. D.1) and only the biased input construction and kernel parameters vary for each problem (described in Sec. D.2).

D.1 FIXED SETTINGS

Across all experiments, the following settings are kept fixed in order to be able to compare the validation performances.

Data: For all subsystems S^c and sub-models M^c ($c = 1, 2, ..., C$) we fix the number of samples to $V_c = 100$ and $n_M = 500$, respectively. All data is generated by a fixed base seed of 2349.
Trials: All experiments report the mean and standard deviation across 5 independent trials where all settings are kept identical except the base seed; this is incremented by the 0-index id of each trial.
Failure probability threshold: The threshold for each validation problem is set such that the ground truth failure probability of the (TPI) output of the last subsystem S^C is at approximately 1%; 1 million samples are used to determine the threshold. The goal of all validation methods across all benchmark problems is to achieve a failure probability as close to 1.0% as possible. It should be noted that in a real-world system, the ground truth failure probability is not available or is poorly approximated through limited samples.
Failure probability grid: The grid of the failure probability optimization is based on the (TPI) output of the last sub-model M^C. It is chosen to cover the entire support of the model output distribution and a significantly large grid region after the threshold τ.
Failure probability Lipschitz constant: Ideally, the Lipschitz-constant should be set according to the smoothness (or Lipschitz constant) of the system's TPI distribution. As this is not available, we used the empirical histogram distribution of simulated TPI outputs as a proxy. More precisely, for each problem a histogram with 100 bins is constructed to compute the

resulting empirical Lipschitz-constant.

Model misfit – Gaussian process: For the cases where we artificially introduce a model misfit in sub-models M^c ($c = 1, 2, ..., C$) (i.e. "Model Misfit" in main Tab. 1) we learn a Gaussian process (GP) for each sub-model M^c to model the input-output mapping of the corresponding subsystem S^c. The goal is to introduce a misfit in a controllable fashion which reflects realistic misfits between real-world systems and simulations. We generate 100 independent training samples for learning the GP (i.e. the training data is not used to solve the validation problem). We use exact GP inference in `GPyTorch` [Gardner et al., 2018] with Radial basis function (RBF) kernels. The `LBFGS-scipy` optimizer is used with a learning rate of $1.0\text{e}{-3}$ for 2000 iterations and 10 restarts with different initializations. All misfit models use the exact same GP settings, therefore the amount of misfit varies across benchmark problems and their individual components.

Biased input: In order to test the limits of the system and validation methods, we specifically use a biased (model) input $q_{\text{biased}-x}$ for each benchmark problem. This bias was constructed such that the TPI model output distribution is shifted farther away from the threshold. Such a bias distribution exploits the weakness of some validation methods (e.g. MCCP and SurrModel) which do not take the input discrepancy between p_x and q_x into account. As a result, using the bias input distribution q_x for the model input yields overly optimistic estimates with a severe underestimation the failure probability (i.e. $F_{\text{max}} < F_{\text{GT}} = 1.0$).

Kernels: All experiments either use a radial basis function (RBF) (squared-exponential) kernel or inverse multiquadratic (IMQ) (also known as rational quadratic) kernel [Gorham and Mackey, 2017]. Both kernels use a jitter of $1\text{e}{-10}$, and the IMQ kernel has a fixed $\alpha = -0.5$. The length scales for both kernels differ for each benchmark problem and channel/signal dimension.

Length scale search: For a given benchmark problem, all length scales across all channels (including each dimension) are optimized to minimize the failure probability F_{max}. It should be note that as F_{max} is a probability, its interpretation is independent of the kernel length scales. As a result, one can perform a kernel length scale search with the objective of minimizing F_{max}. For all problems, we performed a Bayesian optimization search [Fröhlich et al., 2020] where the search space of all parameters are kept large ($[1.0\text{e}{-8}, 5.0\text{e}3]$), except the length scale of the last TPI kernel which depends on the output range of each problem. Furthermore, we only perform the length scale search for the setting "Perfect Input" and "Perfect Model" (i.e. top left quadrant in main Tab. 1). The found length scales are kept fixed for all other settings.

D.2 RELIABILITY BENCHMARK PROBLEMS

The following subsections provide all details for each reliability benchmark problem/dataset used in main Tab. 1.

D.2.1 Controlled Solvers [Sanson et al., 2019]

Components: This problem has 4 components (i.e. solvers): the Sobol function, Ishigami function, and the remaining two are products of polynomial functions and trigonometric functions.

$$f_1 : (x_{1:5}) \mapsto \prod_{k=1}^{5} \frac{|4x_k - 2| + a_k}{1 + a_k} =: x_6 \tag{21}$$

$$f_2 : (x_{6:8}) \mapsto \sin x_6 + 0.7 \sin^2 x_7 + 0.1 x_8^4 \sin x_6 =: x_9 \tag{22}$$

$$f_3 : (x_{9:14}) \mapsto x_{10}^2 \arctan 1 - x_{14} + x_{11} x_{12} x_{13}^3 + 3x_9 =: x_{15} \tag{23}$$

$$f_4 : (x_{15:19}) \mapsto \sin x_{19} x_{18} + x_{15} x_{16} + x_{17}, \tag{24}$$

where $a = (12, 2, 3, 4, 45)$, the $x_6 = f_1(x_{1:5})$, $x_9 = f_2(x_{6:8})$, and $x_{15} = f_3(x_{9:14})$. This problem is also defined in more detail in Sec. 5.5. "Test Case 3" in Sanson et al. [2019] with Fig. 14 depicting the causal graph.

Perfect Input: 16-dimensional input ($x_{\{1,2,...,19\}\setminus\{6,9,15\}}$) sampled from $\mathcal{U}_{[0.0,1.0]}$.

Biased Input: Input signal x_{18} sampled from $\mathcal{U}_{[0.0,0.8]}$ and the remaining inputs ($x_{\{1,2,...,19\}\setminus\{6,9,15\}}$) from $\mathcal{U}_{[0.0,1.0]}$.

Model Misfit: A GP is learned for each component M^c ($c = 1, 2, ..., C$).

Failure probability: The grid range is fixed to $[g_{\text{min}} = -5.0, g_{\text{max}} = 60.0]$ with the threshold at $\tau = 14.51$. A (decreasing) monotonicity constraint is enforced for the range $[\tau - 1.5 * \text{grid-spacing}, g_{\text{max}}$. The Lipschitz constant is set to 0.28.

Kernels: All channels use a RBF kernel with the following length scales: $[1.0\text{e}{-6}, 5.0e01, 1.0\text{e}{-6}, 5.0\text{e}1, 1.0\text{e}{-6}, 1.0\text{e}{-6}, 1.0\text{e}{-6}, 6.397]$ for the channels $[x_{1:5}, x_{6:7}, f_1(\cdot), x_{8:12}, f_2(\cdot), x_{13:16}, f_3(\cdot), f_4(\cdot)]$, respectively.

D.2.2 Chained Solvers [Sanson et al., 2019]

Components: This problem has 2 components (i.e. solvers) forming a composition of two univariate functions f_1 and f_2:

$$f_1 : x \mapsto e^{\sqrt{x}} \sin x + 6e^{-(x-2)^2} + \frac{5}{2} e^{-3(x-1)^2} \tag{25}$$

$$f_2 : x \mapsto \sin x + 0.3x \sin 3.4x + 0.5, \tag{26}$$

where the global output is $f = f_1 \circ f_2$. This problem is also defined in more detail in Sec. "5.1. Test Case 1" in [Sanson et al., 2019] with Fig. 4 plotting the signals.
Perfect Input: Univariate input sampled from $\mathcal{U}_{[0.0,6.0]}$.
Biased Input: Univariate input sampled from a mixture distribution $\alpha\mathcal{U}_{[0.0,6.0]} + (1-\alpha)\mathcal{U}_{[4.0,6.0]}$ with $\alpha = 0.90$. The α controls the trade-off between the biasedness and correctness of the support of the resulting distribution.
Model Misfit: A GP is learned for each component M^c $(c = 1, 2, ..., C)$.
Failure probability: The grid range is fixed to $[g_{\text{min}} = -8.0, g_{\text{max}} = 5.0]$ with the threshold at $\tau = 1.459$. A (decreasing) monotonicity constraint is enforced for the range $[\tau - 1.5 * \text{grid-spacing}, g_{\text{max}}$. The Lipschitz constant is set to 99.0.
Kernels: The 2 single-dimensional input channels and the TPI channel use a RBF and IMQ kernel, respectively, with length scales $[1\text{e}{-8}, 1\text{e}{-8}, 1.218]$.

D.2.3 Borehole [Surjanovic and Bingham, 2023]

Components:

This problem has a single component which models water flow through a borehole:

$$f : (r_w, r, T_u, H_u, T_l, H_l, L, K_w) \mapsto \frac{2\pi T_i (H_u - H_l)}{\ln(r/r_w)(1 + \frac{2LT_u}{\ln(r/r_w) r_w^2 K_w} + \frac{T_u}{T_l})} \tag{27}$$

This problem is also defined in more detail at `https://www.sfu.ca/~ssurjano/borehole.html`. We construct two variants of this problem: `single_borehole` and `compositional_borehole`. The former considers only the function f above, whereas the latter breaks the function up into multiple (5) smaller components:

$$f_1 : (T_u, H_u, H_l) \mapsto 2\pi T_i (H_u - H_l) \tag{28}$$

$$f_2 : (r_w, r, T_u, L, K_w) \mapsto (\frac{2LT_u}{\ln(r/r_w) r_w^2 K_w} \tag{29}$$

$$f_3 : (T_u, T_l) \mapsto \frac{T_u}{T_l} \tag{30}$$

$$f_4 : (r_w, r, y2, y3) \mapsto \ln(\frac{r}{r_w}(1 + y_2 + y_3)) \tag{31}$$

$$f_5 : (y_1, y_4) \mapsto \frac{y_1}{y_2}, \tag{32}$$

where $y_1 = f_1(\cdot)$, $y_2 = f_2(\cdot)$, $y_3 = f_3(\cdot)$, and $y_4 = f_4(\cdot)$.
Perfect Input: The 8 inputs are sampled from distributions described in detail at `https://www.sfu.ca/~ssurjano/borehole.html`. The description also includes the range of all signals in the system.
Biased Input: The sampling range of the H_u signal is modified from $[990, 1110]$ to $[990, 1010]$.
Model Misfit: A GP is learned for each component M^c $(c = 1, 2, ..., C)$.
Failure probability: The grid range is fixed to $[g_{\text{min}} = -35.0, g_{\text{max}} = 600.0]$ with the threshold at $\tau = 157.1$. A (decreasing) monotonicity constraint is enforced for the range $[\tau - 1.5 * \text{grid-spacing}, g_{\text{max}}$. The Lipschitz constant is set to 0.0006.
Kernels: For the `single_borehole`, the RBF kernel is used for both input and output kernels with the following length scales: $[[10.599, 6.587, 24.609, 32.369, 46.431, 23.046, 12.943, 2.734], 23.578]$, respectively. The first kernel has a multi-dimensional length scale; one for each input dimension. For the `compositional_borehole`, the RBF kernel is used for all 8 input channels and a IMQ kernel for the TPI output kernel with length scales: $[5.0\text{e}{+3}, 1.\text{e}{-1}, 3.198\text{e}{+3}, 5.0\text{e}{+3}, 1.0\text{e}{-1}, 5.0\text{e}{+3}, 5.0\text{e}{+3}, 1.0\text{e}{-1}, 5.0\text{e}{+3}, 2.634]$, respectively.

D.2.4 Branin [Surjanovic and Bingham, 2023]

Components: This problem has a single component:

$$f : x_{1:2} \mapsto f_{\text{max}} - a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1-t)\cos x_1 + s, \tag{33}$$

where we the recommended parameters are used: $a = 1$, $b = 5.1/(4\pi^2)$, $c = 5/\pi$, $r = 6$, $s = 10$ and $t = 1/(8\pi)$. In order to map the minimization problem to our maximization setting, we modify the branin function by adding $f_{\text{max}} = 312.0$ and subtracted the original formulation thereof. This problem is also defined in more detail at `https://www.sfu.ca/~ssurjano/branin.html`

We construct two variants of this problem: `single_branin` and `compositional_branin`. The former considers only the function f above, whereas the latter breaks the function up into multiple (3) smaller components:

$$f_1 : x_{1:2} \mapsto (x_2 - bx_1^2 + cx_1 - r)^2 \tag{34}$$

$$f_2 : x_{1:2} \mapsto (1-t)\cos x_1 \tag{35}$$

$$f_3 : x_{3:4} \mapsto f_{\text{max}} - ax_3 + sx_4 + s, \tag{36}$$

where $x_3 = f_1(\cdot)$ and $x_4 = f_2(\cdot)$.

Perfect Input: The 2 inputs are sampled from $\mathcal{U}_{[-5.0,10.0]}$ and $\mathcal{U}_{[0.0,15.0]}$, respectively.

Biased Input: The 2 inputs are sampled from a mixture distribution $\alpha\mathcal{U}_{[-5.0,10.0]} + (1-\alpha)\mathcal{U}_{[8.0,10.0]}$ and $\alpha\mathcal{U}_{[0.0,15.0]} + (1-\alpha)\mathcal{U}_{[12.0,15.0]}$ with $\alpha = 0.10$, respectively. The α controls the trade-off between the biasness and correctness of the support of the resulting distribution.

Model Misfit: A GP is learned for each component M^c $(c = 1, 2, ..., C)$.

Failure probability: The grid range is fixed to $[g_{\text{min}} = -35.0, g_{\text{max}} = 700.0]$ with the threshold at $\tau = 330.82$. A (decreasing) monotonicity constraint is enforced for the range $[\tau - 1.5 * \text{grid-spacing}, g_{\text{max}}$. The Lipschitz constant is set to 0.005.

Kernels: For the `single_branin`, the RBF kernel is used for both input and output kernels with the following length scales: $[0.003, 21.161]$, respectively. For the `compositional_branin`, the IMQ kernel is used for all channels with length scales: $[1\text{e}{-8}, 1\text{e}{-8}, 500.0, 1\text{e}{-8}, 26.064]$.

D.2.5 Four Branch [UQWorld, 2019]

Components: This problem has 4 independent components which form four branches and the final global output takes the minimum of the four component outputs:

$$f_1 : x_{1:2} \mapsto 3 + 0.1(x_1 - x_2)^2 - \frac{x_1 + x_2}{\sqrt{2}} \tag{37}$$

$$f_2 : x_{1:2} \mapsto 3 + 0.1(x_1 - x_2)^2 + \frac{x_1 + x_2}{\sqrt{2}} \tag{38}$$

$$f_3 : x_{1:2} \mapsto (x_1 - x_2) + \frac{p}{\sqrt{2}} \tag{39}$$

$$f_4 : x_{1:2} \mapsto (x_1 - x_2) - \frac{p}{\sqrt{2}} \tag{40}$$

$$f_5 : x_{1:2} \mapsto \min\{f_1(x_{1:2}), f_2(x_{1:2}), f_3(x_{1:2}), f_4(x_{1:2})\} + 10, \tag{41}$$

where $p = 6.0$. This problem is also defined in more detail in UQWorld [2019], where Fig. 1 shows the surface plot of the four branch function. We construct two variants of this problem: `single_four_branch` and `compositional_four_branch`. The former considers only the function f_4 above, whereas the latter breaks the function up into multiple (4) smaller components.

Perfect Input: The 2 inputs are sampled from two normal distributions described as in detail at UQWorld [2019].

Biased Input: The 2 inputs are sampled from a mixture distribution $\alpha\mathcal{N}(0.0, 1.0) + (1-\alpha)\mathcal{N}(0.0, 0.90)$ and $\alpha\mathcal{N}(0.0, 1.0) + (1-\alpha)\mathcal{N}(0.0, 0.9)$ with $\alpha = 0.80$, respectively. The α controls the trade-off between the biasness and correctness of the support of the resulting distribution.

Model Misfit: A GP is learned for each component M^c $(c = 1, 2, ..., C)$.

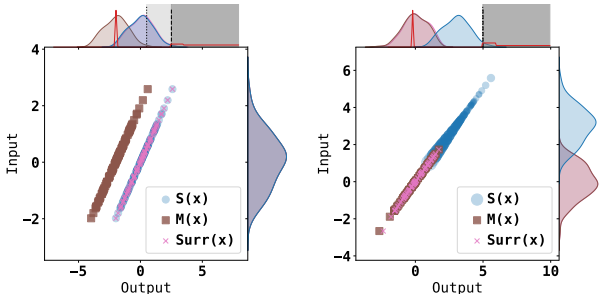

Figure 1: Illustration of `DPBound` (see also Fig. 3 in the main text) and `SurrModel` for a linear mapping between Gaussian signals. **(a)** Model and system are different $S \neq M$, whereas the input distributions are identical $p_x = q_x$. **(b)** The model M is the perfect model, i.e. $S = M$, but input distributions are different. Computed weights α_v (see Eq. (3) in Sec. 3.2) are indicated by the size of markers for $S(x)$ and the worst-case distributions w.r.t. the failure probability are indicated in red. The inputs and outputs of the surrogate model `SurrModel` are shown in pink.

Failure probability: The grid range is fixed to $[g_{\text{min}} = 0.0, g_{\text{max}} = 30.0]$ with the threshold at $\tau = 9.693$. A (decreasing) monotonicity constraint is enforced for the range $[\tau - 1.5 * \text{grid-spacing}, g_{\text{max}}$. The Lipschitz constant is set to 0.005.
Kernels: For the `single_four_branch`, the RBF kernel is used for both input and output kernels with the following length scales: $[[0.201, 0.198], 10.0]$, respectively. For the `compositional_four_branch`, the RBF kernel is used for all channels with length scales: $[[2.018, 1.983], 2.472, 2.374, 2.077, 2.077, 10.0]$.

E ILLUSTRATIONS OF THE METHODS & FURTHER EXPERIMENTS (SEC. 4)

E.1 SURROGATE MODEL `SURRMODEL` IN THE TOY SETTING (SEC. 4.1)

In Sec. 4.1, an illustrative example was used to visualize the two configurations considered in the experimental setup. Here, we additionally analyze the performance of the surrogate model (i.e. the `SurrModel` method from Sec. 3.3) for this single-component linear example under the two configurations.

We illustrated and discussed in Sec. 4.1 how `DPBound` can handle biases in the input distribution, as well as mismatches between the models S and M. On the other hand, the explicit uncertainty estimation with surrogate models fails to handle or detect mismatches in the input distribution, because the estimate of the output distribution arises from surrogate models (albeit learned on the validation data from S) run on the input distribution of M, thereby completely ignoring the real-world input distribution p_x of S. To see this, note that the resulting output distribution (pink crosses) of the surrogate model in Suppl. Fig. 1(a) lies on top of the system output distribution $S(\cdot)$ (which is different from $M(\cdot)$), whereas in Suppl. Fig. 1(b) it basically coincides with the model output distribution $M(\cdot)$ (so that no difference is detected). Consequently, surrogate models can detect differences due to modeling mismatches $M \neq S$, but not between input distributions $q_x \neq p_x$.

E.2 ILLUSTRATION OF SIGNAL AND ERROR PROPAGATION (FOR THE "CHAINED SOLVERS" USECASE, SEC. 4.2)

To illustrate the propagation of signals through the chain of subsystems, Suppl. Fig. 2 shows the propagation of signals through the system (top row), the model chain (middle row), and the surrogate model chain (bottom row, for the `SurrModel` method from Sec. 3.3). This is shown for the "Chained Solvers" use-case in the setting of "Biased Input–Misfit Model" (see Sec. 4.2 and lower right quadrant of the main Tab. 1); we picked this "Chained Solvers" use-case for visualization purposes as it has one-dimensional signals. The mismatches (errors) between those three signals are illustrated in Suppl. Fig. 3.

Suppl. Fig. 3 illustrates the propagation of errors (residuals) between system/model and surrogate model through the components, in the same setting as Suppl. Fig. 2 (described in the previous paragraph). As the `SurrModel` method (Sec. 3.3), starts from the simulation inputs (bottom row of Suppl. Fig. 3), which may be biased w.r.t. the real-world input

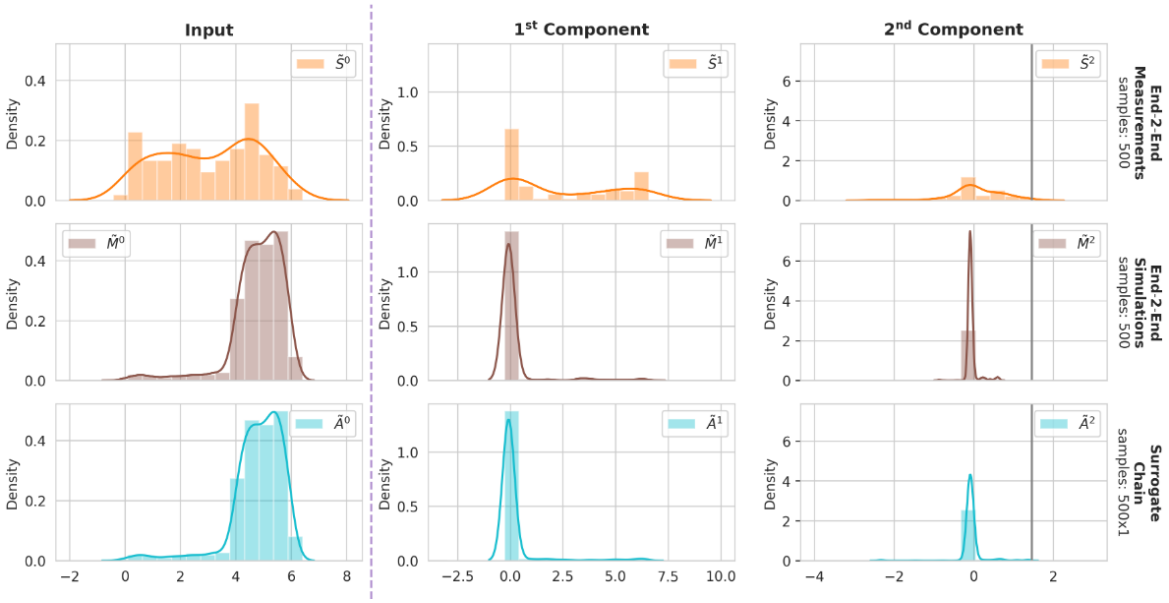

Figure 2: Figure showing the input/output signal distributions for the "Chained Solvers" use-case in the setting "Biased Input–Misfit Model" (cf. main Tab. 1; we chose this use-case for illustration purposes, as its signals are one-dimensional). **Top row:** ground-truth signals (from the system S). **Middle row:** simulation signals (from the model M). **Bottom row:** surrogate model signals (from the model M' in the `SurrModel` method, see Sec. 3.3). **Left column:** input distributions. **Middle column:** output distributions after first component. **Right column:** final TPI distributions.

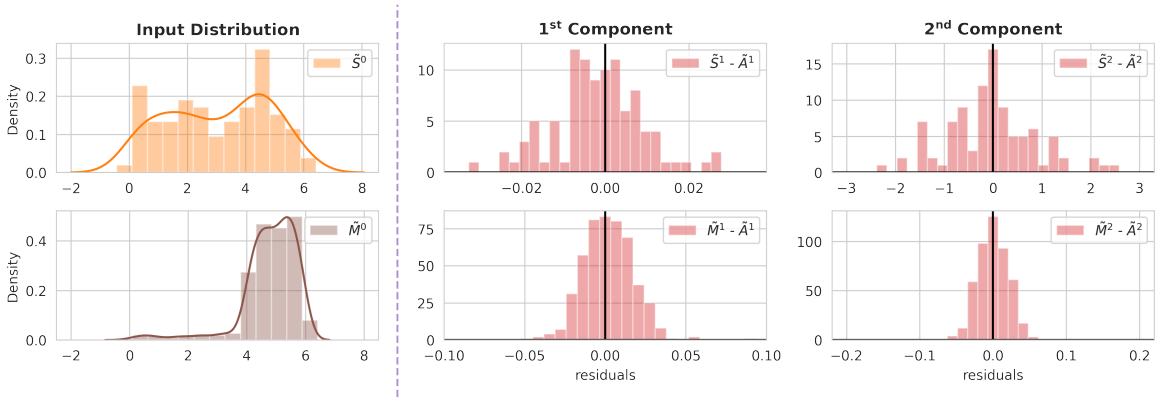

Figure 3: Figure showing the propagation of errors (residuals) of the surrogate model for the "Chained Solvers" use-case in the setting "Biased Input–Misfit Model" (cf. main Tab. 1); see Fig. 2 for the actual signals. **Left column:** real-world input distribution (top) and simulation input distribution (bottom; note that the simulation input distribution is biased). **Top row (2nd and 3rd column):** histograms over residuals (errors) between real system and surrogate model (both starting from the real-world input distribution). **Bottom row (2nd and 3rd column):** histograms over residuals (errors) between simulation model and surrogate model (both starting from the simulation input distribution), akin to what the `SurrModel` method uses (see Eq. (6) in Sec. 3.3).

distribution (top tow), the residuals from Eq. (6) in Sec. 3.3 used by `SurrModel` may be too small (compare lower-right vs. upper-right panel in Suppl. Fig. 3), finally leading to an underestimate of the failure probability by `SurrModel` in Eq. (7) (Sec. 3.3). This failure of the `SurrModel` method is observed in the actual experiments (main Tab. 1 in the main text), especially for the "biased-input" settings.

E.3 DEPENDENCE OF `MCCP` ON ITS CONFIDENCE LEVEL PARAMETER

In Suppl. Tab. 1 we corroborate our conclusions from the experiments in Sec. 4.2 regarding the (in)validness of the `MCCP` method.

Table 1: Ratio of invalid bounds (i.e. bounds below 1%) produced by the `MCCP`-method run with a confidence (CL) parameter of 99% in each of the four simulations configurations, compared (in parentheses) to the ratio for `MCCP` at 95% CL parameter from main Tab. 1. While the ratio of invalid bounds does decrease with the higher CL parameter of 99%, the ratio does not decrease in a proportionate way down to one fifth of the ratio at 95%-CL, especially not for the *Biased Input* settings. The invalidness ratio stays clearly above its 1% validity promise (except in the easy case of *Perfect Input–Perfect Model*). This indicates that `MCCP`'s CL parameter is *not* the main reason for its (high) level of invalidity. The main reason is rather `MCCP`'s ignorance of the system input distribution and of the model misfits (see Sec. 4.2).

`MCCP` at 99% CL (95% CL)	**Perfect Model**	**Misfit Model**
Perfect Input	0% (0%)	5% (22.5%)
Biased Input	47.5% (67.5%)	42.5% (67.5%)

E.4 TIGHTNESS OF THE SDP RELAXATION (LEMMA 1)

Here we investigate experimentally how tight our convex (SDP) relaxation of the nonconvex bound optimization in Eq. (4) from Sec. 3.2 is (see also App. A). For this, we evaluate the *minimum approximation ratio* $\widehat{\gamma} \in [0, 1]$ of the SDP relaxation, as defined below Lemma 1 in App. A, for each of the 440 SDP optimizations required to produce our main results table (Tab. 1 in Sec. 3, which summarizes $8 \cdot 4 \cdot 5 = 160$ validation runs). Note that $\widehat{\gamma} = 1$ would be proof of a perfectly *tight* relaxation, while for example $\widehat{\gamma} = 0.99$ guarantees that the relaxation was tight up to at most 1%. These tightness results are summarized in Suppl. Tab. 2.

Table 2: Frequency of *minimum approximation ratios* $\widehat{\gamma}$ of the SDP relaxations (defined below Lemma 1 in App. A) for the 440 SDP relaxations required to produce Tab. 1 in Sec. 3. Note that the true but unknown approximation ratio γ of each SDP relaxation satisfies $\widehat{\gamma} \leq \gamma \leq 1$. Thus, over all 160 validation tasks from the main Tab. 1, 87.3% of the 440 required SDP bound optimizations are guaranteed to be at least 99%-tight.

minimum approximation ratio $\widehat{\gamma}$	# of SDP optimizations	% of SDP optimizations
$0.99 \leq \widehat{\gamma} \leq 1.0$	384	87.3%
$0.9 \leq \widehat{\gamma} < 0.99$	15	3.4%
$0.1 \leq \widehat{\gamma} < 0.9$	29	6.6%
$0.0 \leq \widehat{\gamma} < 0.1$	12	2.7%
total # of SDP optimizations	440	100%

E.5 COMPUTATIONAL COST & RUNTIME

The computational complexity of the method in terms of the validation data set size(s) and the number of components is discussed at the end of App. A (see also below Eq. (4) in the main text).

Empirically, the runtime required for the 160 validation runs of `DPBound` to produce Tab. 1 on our desktop machine is 2 hours; this runtime is dominated by the semidefinite optimization steps (where we use Eqs. (10–14) in place of Eq. (4), and use the concrete form Eqs. (16–20) in place of Eq. (5)). On those same 160 validation problems, the `MCCP` method takes 2min (as `MCCP` must only propagate model inputs through the given model chain, with no optimizations to do), while `SurrModel` takes 25min (mainly spent on fitting the surrogate Gaussian Process models).

References

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Lukas Fröhlich, Edgar Klenske, Julia Vinogradska, Christian Daniel, and Melanie Zeilinger. Noisy-input entropy search for efficient robust Bayesian optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 2262–2272. PMLR, 2020.

Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Jackson Gorham and Lester Mackey. Measuring sample quality with kernels. In *International Conference on Machine Learning*, pages 1292–1301. PMLR, 2017.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

K. Muandet, K. Fukumizu, B. Sriperumbudur, and B. Schölkopf. Kernel mean embedding of distributions: A review and beyond. *Foundations and Trends in Machine Learning*, 10(1):1–141, 2017.

Jaehyun Park and Stephen Boyd. General heuristics for nonconvex quadratically constrained quadratic programming. *arXiv preprint arXiv:1703.07870*, 2017.

Francois Sanson, Olivier Le Maitre, and Pietro Marco Congedo. Systems of Gaussian process models for directed chains of solvers. *Computer Methods in Applied Mechanics and Engineering*, 352:32–55, 2019.

Sonja Surjanovic and Derek Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved February 13, 2023, from `http://www.sfu.ca/~ssurjano`, 2023.

UQWorld. Four-Branch Function. Retrieved February 13, 2023, from `https://uqworld.org/t/four-branch-function/59`, 2019.