Result-based reinterpretation of the Belle II $B^+ \to K^+\nu\bar{\nu}$ analysis

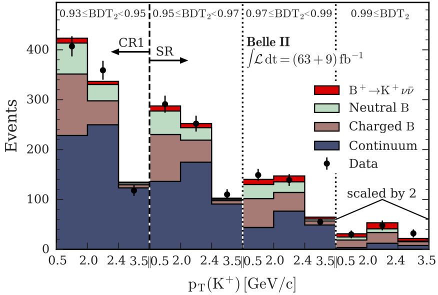
Lorenz Gärtner,
Thomas Kuhr, Danny van Dyk, Lukas Heinrich,
Slavomira Stefkova, Méril Reboud

Excellence Cluster ORIGINS, LMU Munich

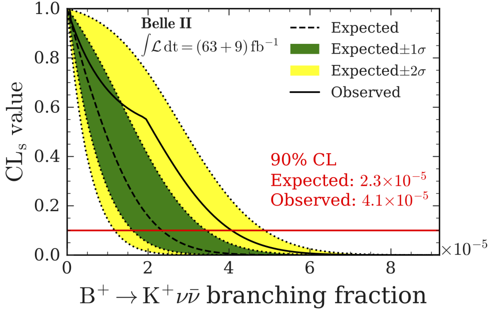
March 3, 2023

The Belle II $B^+ \to K^+ \nu \bar{\nu}$ analysis

[Phys.Rev.Lett.127.181802], HEPData



Fitting in bins of $p_T(K^+) \times BDT_2$



$\mathcal{B}_{B^+ \to K^+ \nu \bar{\nu}} < 4.1 \times 10^{-5}$@90%$CL$

Model dependence: The signal MC is weighted according to the SM expectation.
What would we observe when assuming BSM physics?

(B)SM theory predictions

- We can capture all (B)SM physics within 3 effective Wilson coefficients*

$$C_{VL} + C_{VR}, C_{SL} + C_{SR}, C_{TL}$$

github.com/eos/eos/

*$C_{VL} = C_{VL}^{SM} + C_{VL}^{NP}$

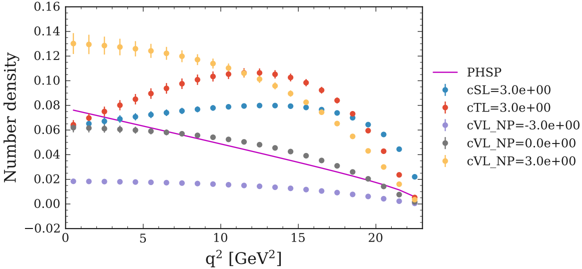
(B)SM theory predictions

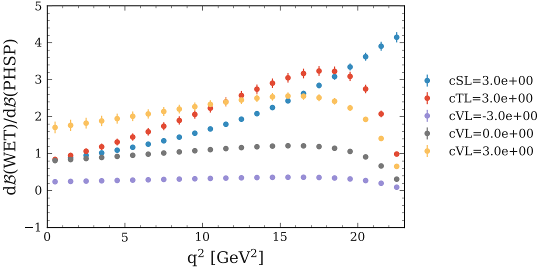
- We can capture all (B)SM physics within 3 effective Wilson coefficients*

$$C_{VL} + C_{VR}, C_{SL} + C_{SR}, C_{TL}$$

- The expected differential branching ratio is a function of the dineutrino invariant mass squared q^2 only.

github.com/eos/eos/





$^*C_{VL} = C_{VL}^{SM} + C_{VL}^{NP}$

Reweighting Approach

Given the pyhf input `json` of the analysis, we only need to know how to reweight the signal.

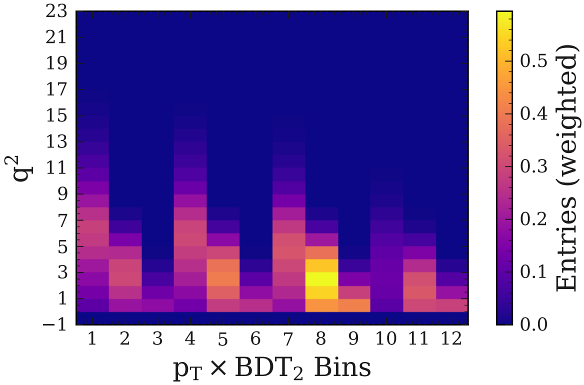
Additional dimension

- 3d binning:

$$\underbrace{p_T \times BDT_2}_{\text{analysis binning}} \times \underbrace{q^2}_{\text{new}}$$

- Apply weights in q^2 bins and resum

+ Good accuracy with sufficient q^2 bins
+ Very versatile
+ Easily publishable



pyhf solutions

1. Patch input `json`

For our signal contribution:

1. Pick a theory model.
2. Calculate the new signal expectations in the analysis bins.
3. Adapt all modifiers that interpolate between absolute bin values, eg. the correlated shape (histosys) modifier.
4. Given the original pyhf input `json`, we can just replace the old signal with the new one using `jsonpatch`.

Docs: jsonpatch or pyhf implementation

Code

```
>>> patch = JsonPatch([
...     {'op': 'add', 'path': '/foo', 'value': 'bar'},
...     {'op': 'add', 'path': '/baz', 'value': [1, 2, 3]},
...     {'op': 'remove', 'path': '/baz/1'},
...     {'op': 'test', 'path': '/baz', 'value': [1, 3]},
...     {'op': 'replace', 'path': '/baz/0', 'value': 42},
...     {'op': 'remove', 'path': '/baz/1'},
... ])
>>> doc = {}
>>> result = patch.apply(doc)
>>> result
{'foo': 'bar', 'baz': [42]}
```

2. Custom modifier

Instead of patching for each new theory model, we can write a modifier that does this for us (PR#1991).

1. Build a function that calculates modifications from a set of nuisance parameters.
2. Decide what properties do you want your modifier to have:
   - Constraint type: Gaussian/Poisson/unconstrained
   - Action type: addition/multiplication
   - Initialization parameters
   - Bounds
   - ...

In our case, Wilson coefficients flow into the likelihood as nuisance parameters.

2. Custom modifier

Instead of patching for each new theory model, we can write a modifier that does this for us (PR#1991).

1. Build a function that calculates modifications from a set of nuisance parameters.
2. Decide what properties do you want your modifier to have:
   - Constraint type: Gaussian/Poisson/unconstrained
   - Action type: addition/multiplication
   - Initialization parameters
   - Bounds
   - ...

In our case, Wilson coefficients flow into the likelihood as nuisance parameters.

Custom modifier implementation

```
def custom_modifier(...):
    def make_function(function_name, ...):
        # adapt pre-defined modification function to binning
        ...
    def _allocate_new_param(p):
        # chose modifier specifications
        return {
            'paramset_type': 'unconstrained',
            'n_parameters': 1,
            'inits': p['inits'],
            'bounds': p['bounds'],
            ...
        }
    class _builder:
        #get modifier input from json
        ...
    class _applier:
        # actually apply our change
        name = 'customfunc' # modifier type ("histosys")
        op_code = 'addition' # chose the modifier action
        ...
        def apply(...):
            # apply custom function to specified bins
            ...
    ...
```

Custom modifier – example

Let us add a simple Gaussian modifier to an empty model.

Input json

```
...
{'name': 'signal',
 'data': [0.,0.,0.,0.,0.,0.,0.,0.,0.,0.],
 'modifiers': [
    {'name': 'gauss_modifier',
     'type': 'customfunc',
     'data':
        {'expr': 'gauss',
         'bins': [-5.,-4.,-3.,-2.,-1.,0.,1.,2.,3.,4.,5.]
        }
    }
 ]
}
...
```

Custom modifier – example

Let us add a simple Gaussian modifier to an empty model.

Input json

```
...
{'name': 'signal',
 'data': [0.,0.,0.,0.,0.,0.,0.,0.,0.,0.],
 'modifiers': [
    {'name': 'gauss_modifier',
     'type': 'customfunc',
     'data':
        {'expr': 'gauss',
         'bins': [-5.,-4.,-3.,-2.,-1.,0.,1.,2.,3.,4.,5.]
        }
    }
 ]
}
...
```

Custom function

```
def gauss(pars):
    def func(xi):
        mean, sigma = pars['mean'], pars['sigma']
        return 1/(np.sqrt(2*np.pi)*sigma) *
        ↪   np.exp(-(xi-mean)**2/(2*sigma**2))
    return func
```

Custom modifier – example

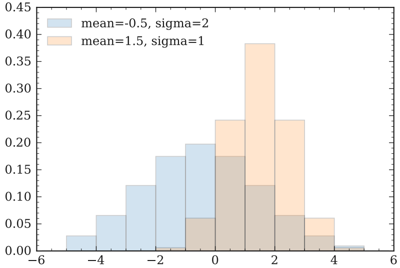
Let us add a simple Gaussian modifier to an empty model.

Input json

```
...
{'name': 'signal',
 'data': [0.,0.,0.,0.,0.,0.,0.,0.,0.,0.],
 'modifiers': [
    {'name': 'gauss_modifier',
     'type': 'customfunc',
     'data':
        {'expr': 'gauss',
         'bins': [-5.,-4.,-3.,-2.,-1.,0.,1.,2.,3.,4.,5.]
        }
    }
 ]
}
...
```

Custom function

```
def gauss(pars):
    def func(xi):
        mean, sigma = pars['mean'], pars['sigma']
        return 1/(np.sqrt(2*np.pi)*sigma) *
        ↪   np.exp(-(xi-mean)**2/(2*sigma**2))
    return func
```

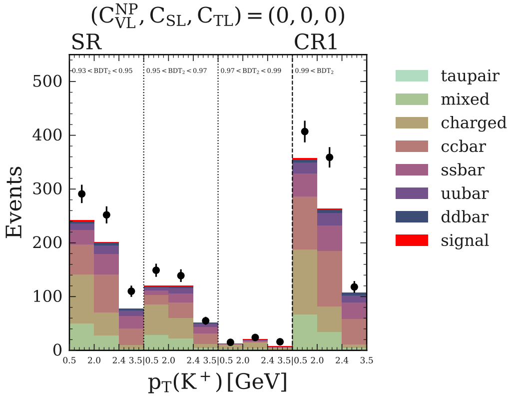


Results

Expected yields

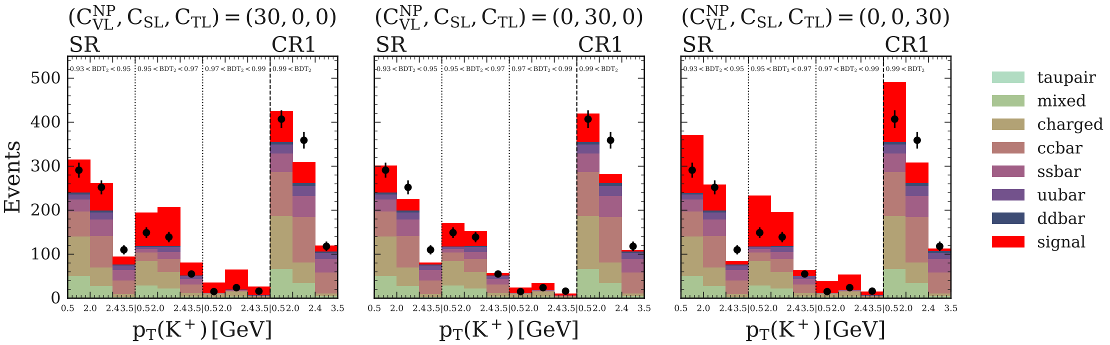
As a first application, we can investigate how the signal contribution changes for different models.
What do we expect?

- The contributions from $C_{SL} + C_{SR}$ and C_{TL} peak at larger values of q^2.
- If their contributions are large, we expect the $p_T(K^+)$ distribution to peak at lower values.
- The tensor contribution is larger than the scalar one, given $C_{SL} = C_{TL}$.



These are the nominal bin entries (pre-fit), according to the SM.

Expected yields



Taking our models to the extreme, we indeed see all the mentioned points confirmed.

Wilson coefficient exclusion

Confidence Limits (CLs)

What we really want, is to confine the theory parameter space.
Hence, we want to do a hypothesis test with $\mu = 1$ in the space of Wilson coefficients.

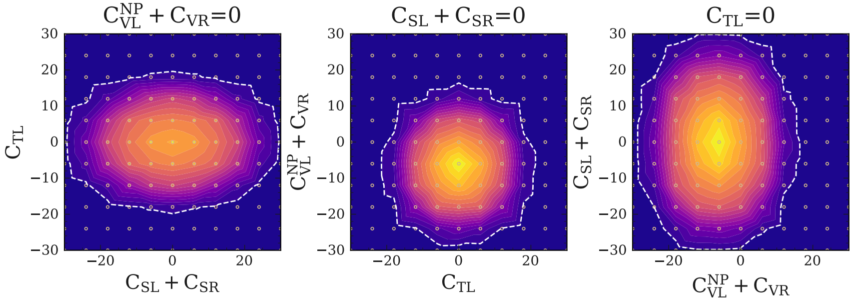
Hypothesis test

```
cls_obs = pyhf.infer.hypotest(
    1,                                # fix mu=1
    data,                             # the observed data points
    model,                            # pyhf model
    init_pars,                        # set the Wilson coefficients
    pdf.config.suggested_bounds(),    # default bounds
    fixed_params=fixed                # fix the Wilson coefficients in the fit
)
```

Wilson coefficient exclusion

Confidence Limits (CLs)

What we really want, is to confine the theory parameter space.
Hence, we want to do a hypothesis test with $\mu = 1$ in the space of Wilson coefficients.



To find an exclusion range, we can look at the contour (dashed line) of $CL_s = 0.05$ in the grid of the 2 parameters.

Fitting Wilson coefficients

Another neat option we have is to fit Wilson coefficients, when using the custom modifier method. To do fitting, our modifierr function should be FAST.

Max. likelihood fit

```
best_fit = pyhf.infer.mle.fit(
    data,
    pdf,
    pdf.config.suggested_init(),
    pdf.config.suggested_bounds(),
    fixed_params=fixed,             # fix mu=1, to avoid
    ↪ interference with Wilson coefficients
    return_uncertainties=True)      # return fit
    ↪ uncertainties (no fit uncertainties with scipy
    ↪ optimizer, but eg. minuit works)
```

Fitting Wilson coefficients

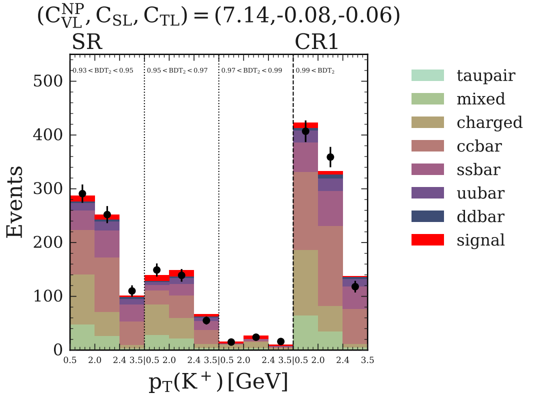
Another neat option we have is to fit Wilson coefficients, when using the custom modifier method. To do fitting, our modifierr function should be FAST.

Max. likelihood fit

```
best_fit = pyhf.infer.mle.fit(
    data,
    pdf,
    pdf.config.suggested_init(),
    pdf.config.suggested_bounds(),
    fixed_params=fixed,             # fix mu=1, to avoid
 ↪  interference with Wilson coefficients
    return_uncertainties=True)      # return fit
 ↪  uncertainties (no fit uncertainties with scipy
 ↪  optimizer, but eg. minuit works)
```

Best fit Wilson coefficients

$$C_{VL}^{NP} + C_{VR} = 7.143 \pm 5.072$$
$$C_{SL} + C_{SR} = -0.075 \pm 44.055$$
$$C_{TL} = -0.060 \pm 21.509$$



Summary

- The $B^+ \to K^+\nu\bar{\nu}$ analysis assumes a SM signal distribution.
- To remove this dependence, we make the likelihood a function of Wilson coefficients.
- pyhf delivers (at least) two solutions for this:
  - Patch the signal region.
  - Implement a custom modifier.
- This easily enables us to
  - Perform a scan over the theory space to check for exclusion.
  - Fit with free Wilson coefficients to check for most likely points in theory space.

Thank you!

lorenz.gaertner@physik.uni-muenchen.de

pyhf

For observed event counts $\boldsymbol{n}$ the likelihood function is composed of

$$L(\boldsymbol{n}, \boldsymbol{a} \mid \boldsymbol{\eta}, \boldsymbol{\chi}) = \underbrace{\prod_{c \in \text{ channels}} \prod_{b \in \text{ bins}} \text{Pois}\left(n_{cb} \mid \nu_{cb}(\boldsymbol{\eta}, \boldsymbol{\chi})\right)}_{\substack{\text{Simultaneous measurement} \\ \text{of multiple channels}}} \quad \underbrace{\prod_{\chi \in \boldsymbol{\chi}} c_{\chi}\left(a_{\chi} \mid \chi\right)}_{\substack{\text{constraint terms} \\ \text{for "auxiliary measurements"}}}$$

with free and constrained parameters $\boldsymbol{\eta}, \boldsymbol{\chi}$, respectively,

$$L(\boldsymbol{x} \mid \phi) = L(\boldsymbol{x} \mid \overbrace{\boldsymbol{\eta}}^{\text{free}}, \underbrace{\boldsymbol{\chi}}_{\text{constrained}}) = f(\boldsymbol{x} \mid \overbrace{\boldsymbol{\psi}}^{\text{parameters of interest}}, \underbrace{\boldsymbol{\theta}}_{\text{nuisance parameters}})$$

The auxiliary measurements $\boldsymbol{a}$ are a frequentist approach to count modification.
The expected number of events for each channel and in each bin is

$$\nu_{cb}(\phi) = \sum_{s \in \text{ samples}} \nu_{scb}(\boldsymbol{\eta}, \boldsymbol{\chi}) = \sum_{s \in \text{ samples}} \underbrace{\prod_{\kappa \in \boldsymbol{\kappa}} \kappa_{scb}(\boldsymbol{\eta}, \boldsymbol{\chi})}_{\text{multiplicative modifiers}} \left(\nu_{scb}^{0}(\boldsymbol{\eta}, \boldsymbol{\chi}) + \underbrace{\sum_{\Delta \in \boldsymbol{\Delta}} \Delta_{scb}(\boldsymbol{\eta}, \boldsymbol{\chi})}_{\text{additive modifiers}}\right).$$

Modifiers and constraints

Description	Modification	Constraint Term c_χ	Input
Uncorrelated Shape	$\kappa_{scb}(\gamma_b) = \gamma_b$	$\prod_b \mathrm{Pois}\left(r_b = \sigma_b^{-2} \middle\vert\, \rho_b = \sigma_b^{-2}\gamma_b\right)$	σ_b
Correlated Shape	$\Delta_{scb}(\alpha) = f_p\left(\alpha \middle\vert\, \Delta_{scb,\alpha=-1}, \Delta_{scb,\alpha=1}\right)$	$\mathrm{Gaus}\left(a = 0 \middle\vert\, \alpha, \sigma = 1\right)$	$\Delta_{scb,\alpha=\pm1}$
Normalisation Unc.	$\kappa_{scb}(\alpha) = g_p\left(\alpha \middle\vert\, \kappa_{scb,\alpha=-1}, \kappa_{scb,\alpha=1}\right)$	$\mathrm{Gaus}\left(a = 0 \middle\vert\, \alpha, \sigma = 1\right)$	$\kappa_{scb,\alpha=\pm1}$
MC Stat. Uncertainty	$\kappa_{scb}(\gamma_b) = \gamma_b$	$\prod_b \mathrm{Gaus}\left(a_{\gamma_b} = 1 \middle\vert\, \gamma_b, \delta_b\right)$	$\delta_b^2 = \sum_s \delta_{sb}^2$
Luminosity	$\kappa_{scb}(\lambda) = \lambda$	$\mathrm{Gaus}\left(l = \lambda_0 \middle\vert\, \lambda, \sigma_\lambda\right)$	$\lambda_0, \sigma_\lambda$
Normalisation	$\kappa_{scb}(\mu_b) = \mu_b$		
Data-driven Shape	$\kappa_{scb}(\gamma_b) = \gamma_b$		

Parameter space selection

The definition of Wilson coefficients in `arXiv:2111.04327 [hep-ph]` compared to the values used in EOS are

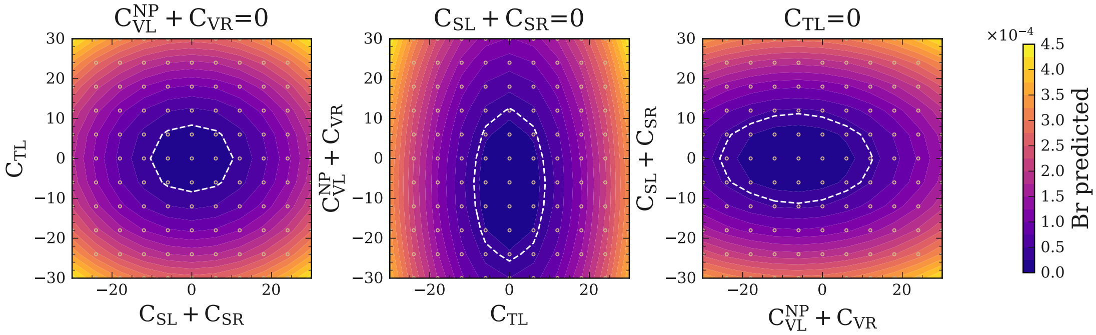
$$C_{paper} = -\frac{4G_F}{\sqrt{2}}\frac{\alpha}{2\pi}V_{ts}^*V_{tb}\left(\frac{X}{\sin^2\theta_W}\right) = -\frac{4G_F}{\sqrt{2}}\frac{\alpha}{2\pi}V_{ts}^*V_{tb}C_{EOS} \approx \frac{1}{615TeV^2}C_{EOS}.$$

We get a rough estimate of the parameter space from `arXiv:2111.04327 [hep-ph]`:

Operator	Value (paper) $[\mathrm{TeV}^{-2}]$	Value (EOS)	NP scale $[\mathrm{TeV}]$	Observable
$\mathcal{O}_{\nu d,\alpha\alpha sb}^{\mathrm{VLL,NP}}$	0.028	17.2	6	$B \to K^*\nu\nu$
$\mathcal{O}_{\nu d,\alpha\alpha sb}^{\mathrm{VLR}}$	0.021	12.9	7	$B \to K\nu\nu$
$\mathcal{O}_{\nu d,\gamma\delta sb}^{\mathrm{VLL}}$	0.014	8.61	9	$B \to K^*\nu\nu$
$\mathcal{O}_{\nu d,\gamma\gamma sb}^{\mathrm{SLL}}$	0.012	7.38	10	$B \to K^{(*)}\nu\nu$
$\mathcal{O}_{\nu d,\gamma\delta sb}^{\mathrm{SLL}}$	0.009	5.54	9	$B \to K^*\nu\nu$
$\mathcal{O}_{\nu d,\gamma\delta sb}^{\mathrm{TLL}}$	0.002	1.23	9	$B \to K^*\nu\nu$

Hence, choosing an upper bound of $C_{EOS} \leq 30$ completely covers branching ratio values of up to $Br \leq 2.5 \times 10^{-4}$

Predicted branching ratio



This plot shows the theoretically predicted branching ratio as a function of Wilson coefficients (WCs). The dashed line corresponds to the upper limit $Br \leq 4.1 \times 10^{-5}$.