Tracking efficiency uncertainty for the STAR TPC

Dmitry Kalinkin[1,2]

[1]Indiana University Center for Exploration of Energy and Matter, Bloomington, Indiana, USA
[2]Brookhaven National Laboratory, Upton, New York, USA

Draft 2

Revision History

Revision	Date	Author(s)	Description
Draft 1	December 18, 2019	D. K.	created
Draft 2	May 13, 2020	D. K.	

Contents

1 Introduction

Most experimental measurements at the STAR detector will rely on information from the Time Projection Chamber (TPC) to reconstruct charged particles. Many analyses will rely on the detector simulation built into the STAR analysis framework to correct for the detector effects such as the tracking efficiency and the tracking resolution. These corrections come with a systematic uncertainty associated with possible inconsistencies between responses of the simulated detector model and the real STAR detector. Systematic uncertainty on tracking efficiency will often make one of the leading contributions to the uncertainty for measurements that use the TPC. For those measurements that look only at the charged particles, it will often make the leading contribution.

To improve the precision of a measurement one could try reducing this uncertainty by choosing a different physics observable that would be less dependent on the tracking efficiency (e.g. one for which the tracking efficiency cancels to some extent). The other way would be to work on improving the simulation to better reproduce the detector effects. Such work has been performed over the years, but no new estimate for the tracking efficiency uncertainty has been made. We believe that the currently used outdated number may be producing overestimated uncertainties. This study is aimed at providing a new estimate, which, hopefully, will be smaller than the current one.

2 Prior work

2.1 STAR NIM TPC paper (2003)

The original STAR TPC paper [1] describes a study where tracking efficiency is measured by embedding simulated tracks into Au + Au events. The systematic for this measurement is quoted as 6%, but the exact procedure for determining this number is not described [There should be an analysis note?].

2.2 Embedding in jets (2006)

Another study [2] with a primary focus on the hadron jets was done on the year 2006 data for $p + p$ collisions at $\sqrt{s} = 200$. For this analysis some simulated charged pions (π^+ and π^-) were embedded into the data sample, one pion was used per each event. A very similar procedure was also performed for a Monte Carlo (MC) simulation sample. This MC was "pure" in a sense that, unlike the conventional STAR "embedding" simulation, the event-by-event mixing of zero-bias data into the simulated events was not performed. The result for the tracking efficiency for particles outside of jet cones with $0.5 \text{ GeV} < p_T < 5 \text{ GeV}$ and $|\eta| < 1$ was $80.0 \pm 0.4\%$ for the data sample and $86.1 \pm 0.5\%$ for the MC sample [2, Table VI].

Embedding of pions inside the cones of the reconstructed jets was also studied and a similar $\sim 6\%$ discrepancy between data and MC was observed. The result for the tracking efficiency for pions with $p_T > 0.2$ GeV embedded inside jets with $|\eta_{\text{det}}| < 0.9$ was $79.5 \pm 0.4\%$ in data and $85.4 \pm 0.5\%$ in MC [2, Table IV], but after the cuts were "tightened" to require pions with $p_T > 0.5$ GeV and $|\eta| < 1$ the efficiencies went up by $\sim 3\%$ to $83.0 \pm 0.5\%$ and $88.3 \pm 0.6\%$ in data and MC samples respectively [2, Table V]. The increase in efficiency is not surprising, given that the efficiency monotonically grows as a function of embedded particle p_T in the range from 0.2 to 0.5. An interesting effect is that the "tighter" cuts are in fact identical to the cuts that were used for the previously mentioned tracking efficiency about the jet cones, yet the numbers disagree by 3% with an enhancement for tracks inside jets.

The thesis also describes how individual acceptance rates for cuts such as a cut on DCA, a cut on the number of fit points and a cut on the fraction of possible fit points were all consistently lower in the data than in the simulation. The cumulative effect of this amounted to a 2.7% difference in tracking efficiencies. This discrepancy was attributed to the pile-up effects that are not reproduced by the "pure" MC and was subtracted from the 5.9% discrepancy to arrive at the resulting systematic uncertainty of 3.3%. Following the logic of this procedure, it would make sense to use the latter number only for an analysis that relies on the embedded simulation or any other kind of simulation that includes reliable facilities to account for the effects of the pile-up.

The source of the remaining discrepancy of 3.3% remains unexplained by the thesis. It is worth pointing out that this discrepancy comes from a difference in tracking of *simulated* tracks, and, although the simulations are slightly different,[1] they should still produce tracks of a very similar quality, so the efficiencies should be similar between the two datasets. Investigating this question was one of the initial inspirations for doing this study.

3 Tracking with the STAR TPC

3.1 TPC detector

The TPC can be roughly described as a barrel 4.2 meters long and 4 meters in diameter filled with a gas. Whenever a charged particle passes the gas volume of the TPC it will collide with the gas particles and create ionization. A uniform electric field created inside the TPC makes electrons from the ionization drift from the center towards the sides of the barrel. In the end, they approach the anode wires where they create an avalanche ionization, so that the amplitude of the image charge from that ionization can be read out from the sensitive pads. Measurement of the drift time is used to determine the distance from initial ionization to the TPC endcaps, thus providing the z coordinate measurement. The pads are arranged in 45 rows so that the position of a given track can be potentially constrained in 45 positions (at different "reconstruction layers").

[1] It appears that an "ADC level" mixing was used for the data, but not in the MC, which also causes inconsistencies between the procedures for vertex origin positioning.

3.2 Track reconstruction

The information from TPC coming in the form of drift times and charges sensed by the TPC pads in their respective positions needs to be converted to information about physical tracks such as their position, momentum vector and energy deposition dE/dx. The reconstruction process consists of several steps:

1. Signals from individual pads within a single pad row that meet a certain criteria are converted to so-called TPC "hits". The hits are characterized by the id of the pad row (layer) they belong to, their position (which is averaged from up to three pads), the magnitude of the detected ionization. This procedure is applied "online" on all of the collected data (e.g. for `st_physics` stream) to reduce the amount of information that needs to be stored. The original ADC waveforms from the pads are still recorded for some subset of the collected data (e.g. for `st_physics_adc` and `st_zerobias_adc` streams). A description of an early "hit finder" algorithm can be found in [3].

2. The positions of the hits are corrected for distortions due to the E and B field nonhomogeneities, misalignments and noncollinearities[2] in those two fields. These include contribution of the time-dependent E field from the ion space charge and from ions leaking (predominantly around the gating grid) back into the TPC drift volume.[3]

3. Hits are combined into the "global tracks". This is done using a track reconstruction algorithm. The STAR framework currently implements two different algorithms for this called "`Sti`" and "`StiCA`" (present study looks at the `Sti` only).

4. If global tracks in the event are pointing to the same location (usually, along the beam line) this location is taken as a "primary vertex" seed. Then the information from the global tracks pointing to the seed along with information from the fast detectors is used to give vertices "rank" which rates our confidence that this vertex was a result of a beam-beam collision, and that it occured in the same bunch crossing as the recorded event. The global tracks are refit again using the vertex position as an additional point on the track, the momentum of the track is corrected for the measured energy loss, the resulting track with slightly improved parameters is referred to as the "primary track". In STAR, a vertex finding algorithm called "PPV" is usually used for proton-proton collisions and "MinutVF" is used for collisions involving the heavy ions[7].

3.3 Simulation

Simulation of tracking inside of the TPC detector is using GEANT to model interactions between relativistic charged particles and the gas of the TPC. This produces a set of "GEANT hits" that represent the charge clusters inside the TPC. The simulation for transport in the electromagnetic field, avalanche process and readout by electronics is then performed using an `StTpcRsMaker` code. A typical simulation in STAR will also embed the resulting simulated event into an event from the zero-bias data. This involves adding ADC waveforms in 512 timebins for each of the $136,608$ pads ($175,440$ pads after the iTPC upgrade). The resulting information is then processed using the "offline" hit reconstruction code and the resulting hits are used for the track reconstruction.

3.4 Tracks in physics analysis

The primary vertices and associated primary tracks coming out from the standard reconstruction procedure will contain pile-up contributions. The pile-up vertices can be effectively removed by using only vertices with positive ranks and picking the one with the highest rank if there is more than one positively ranking vertex. Chosen vertex of interest may still have picked up some tracks from the pile-up interactions. Some relevant tracks might have been misreconstructed due to distortions and need to be also discarded. The criteria by which undesirable tracks are identified are referred to as "track QA cuts". In this study a specific set of track QA cuts is considered, a set of cuts commonly used by Spin PWG for jet analyses in proton-proton collisions. The primary tracks have to satisfy a following set of requirements:

[2] Contributions to $E \times B$ create axial shifts due to the Lorentz force

[3] See [4] for details about the correction, [5] for relevant formalism, and [6] for illustrations of the grid leak.

- *A number of hits (layers) used for a track reconstruction must exceed* 12 – needed to exclude short tracks that can possibly have their parameters measured less precisely.
- *A track must have at least one hit in the outer TPC*[8]
- *The ratio of the number of hits to the number of possible hits must exceed* 0.51 – prevents double counting of tracks that had their different segments reconstructed as independent tracks
- *Distance of the Closest Approach (DCA) of the original global track to the vertex must exceed a value given by a formula:*

$$\mathrm{DCA}^{\max} = \begin{cases} 2 \text{ cm} & \text{if track } p_T < 0.5 \text{ GeV} \\ (2.5 \text{ cm} - p_T \cdot (1 \text{ cm/GeV})) & \text{if } 0.5 \text{ GeV} \leq \text{track } p_T < 1.5 \text{ GeV} \\ 1 \text{ cm} & \text{if } 1.5 \text{ GeV} \leq \text{track } p_T \end{cases}$$

  – removes tracks from secondary decays
- *Track* p_T *must be greater than* 0.2 *GeV*
- *Track* η *must be in the interval* $[-2.5, 2.5]$

Some analyses with a focus on forward jets using the endcap electromagnetic calorimeter (EEMC) will relax the absolute number of hits cut to require only 5 hits on track and doesn't require a hit in the outer TPC while keeping rest of the cuts the same. This set of cuts will be referred to as "forward track QA cuts".

Analyses that rely on dE/dx measurement may require number of points used for dE/dx estimation, $NHits_{dE/dx}$, to be 15 or more.

4 Data and simulation samples

This study looks at the data collected during $p + p$ collisions during run the year 2012. Specifically, the following runs were studied:

- 13059007 (a run at the beginning of the fill 16480, $\sqrt{s} = 200$ GeV)
- 13059025 (a run at the end of the fill 16480, $\sqrt{s} = 200$ GeV)
- 13104003 (a run at the beginning of the fill 16716, $\sqrt{s} = 510$ GeV)

The data recorded during these runs in `st_physics_adc` stream contains events that fired physics collisions triggers, we will refer to that data as just "data". Another subset of data recorded in `st_zerobias_adc` stream has events from the "zero-bias" trigger and those data are used to produce the "embedding" simulation.

4.1 Procedure for the data production

The official production of the pp200 physics data was originally done using the `SL12d` version of the STAR library with the following options for the Big Full Chain:

```
DbV20130212 pp2012b AgML mtdDat btof fmsDat VFPPVnoCTB useBTOF4Vtx beamline
BEmcChkStat Corr4 OSpaceZ2 OGridLeak3D -hitfilt
```

Whereas the official production of the pp510 data is using `SL13b` with options:

```
DbV20130502 pp2012b AgML mtdDat btof fmsDat VFPPVnoCTB beamline BEmcChkStat Corr4
OSpaceZ2 OGridLeak3D -hitfilt
```

For this analysis, the data production for both collisions energies was performed from scratch using the available `SL13b_embed` library. This was done using the respective options from the official production plus the `TpxRaw` and `TpxClu` options to enable the offline hit reconstruction. Offline hit reconstruction is needed for consistency with single track embedding.

The custom data production was verified against the official files on the event-by-event basis. Multiplicities of primary tracks were compared for each event. Kolmogorov–Smirnov metric was calculated

	data	embedding	pure MC
Standard	N/A	160	N/A
With single track embedded	20	160	N/A

Table 2: Reuse factors for the data of the run 13059025 ($\sqrt{s} = 200$ GeV).

	data	embedding	pure MC
Standard	N/A	40	N/A
With single track embedded	20	40	N/A

Table 3: Reuse factors for the data of the run 13104003 ($\sqrt{s} = 510$ GeV).

for each event between the distributions of the track p_T, η and φ in the custom and the official productions, the majority of resulting values were close to zero suggesting that the distributions were very close.

An embedding simulation sample was produced by embedding Pythia events into the zero-bias data collected during the studied runs. The z coordinates of the vertices were sampled from a normal distribution with $\mu = -2$ cm and $\sigma = 26.20$ cm that matches the width of the distribution for VPDMB-triggered events. The x and y were set to the beamline position. Events from the zero-bias sample had to be reused multiple times (see section 4.1) to provide extra statistics. Different random vertex positions and random simulated events were used for each repetition. The Pythia 6.4.28 was set up without any cuts on partonic $\hat{p}_T$ to avoid the need to apply any additional reweighting in the analysis.

A "pure MC" simulation sample was produced by using the same code as for producing embedding with an additional step that removes TPC information coming from the zero-bias files before it can get mixed with the simulated Pythia event and the single track. This procedure ensures that all of the database parameters are initialized in the same way as they would do for embedding. The "pure MC" and embedding simulations were simulating the same set of physics events.

4.2 Procedure for a production with a single embedded particle

The samples with single particles embedded into the data, into the embedding and into the "pure MC" were produced. First, a single particle (in our case π^+ or e^+) was thrown into the detector simulation. The origin vertex of the particle was placed with the same (x, y, z) coordinate as the highest-ranking vertex in the same event reconstructed in the original event. The pseudorapidities for the simulated tracks were chosen randomly from a uniform distribution in the interval $[-1.5, 1.5]$, the transverse momenta of the tracks were chosen from a uniform distribution in the range [0.2 GeV, 2 GeV]. Finally, the original samples (data, embedding and pure MC) were produced from scratch in the same way as described in section 4.1 with one additional step where the TPC ADC response of the simulated particle was added to the event. The response to the simulated particle in all other detectors (like BEMC, BTOF and EEMC) was not accounted for by this procedure. The `st_physics` data was reused multiple times with different simulated particles embedded as indicated in the second row of section 4.1.

4.3 Data-simulation general QA comparisons

A general check was performed to ensure that the generated simulation samples are consistent with the data.

4.3.1 Vertex level quantities

The distributions of the z coordinates of the reconstructed vertices can be seen on figs. 1 and 2. The distributions are in a reasonable agreement considering that no additional reweighting was performed on them.

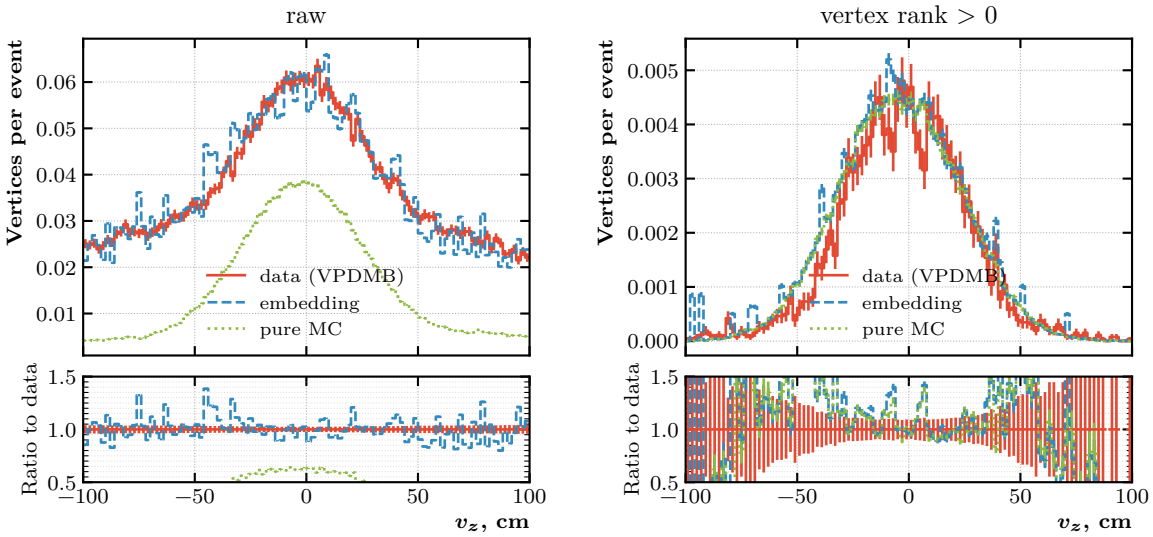

Figure 1: Comparison of the vertex distributions between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of all events in a given sample with no normalization to the bin width. The peaks on the raw distribution for embedding are due to the zero-bias data reuse.

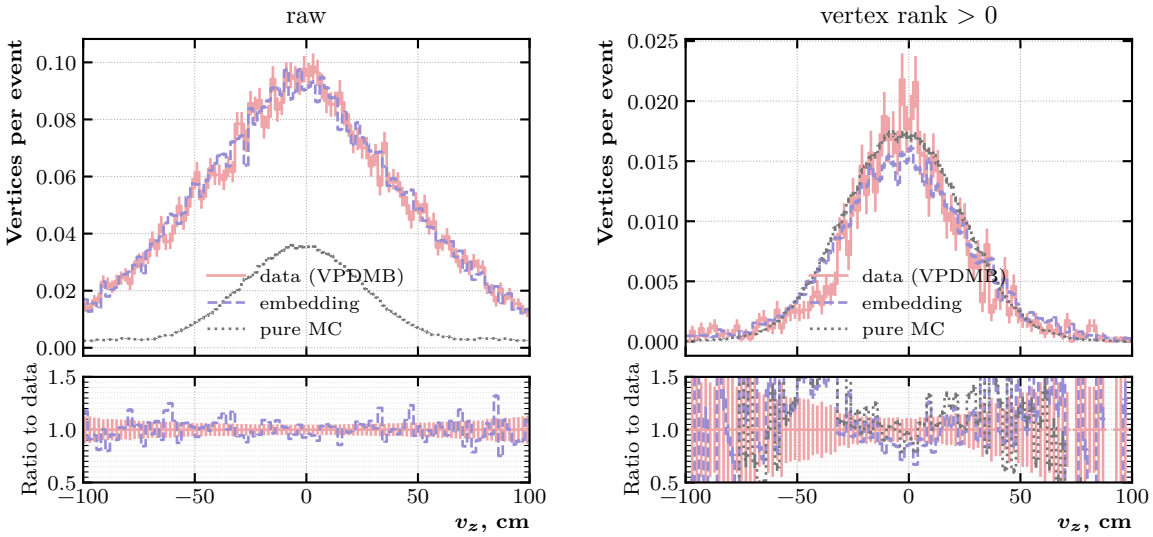

Figure 2: Comparison of the vertex distributions between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of all events in a given sample with no normalization to the bin width.

4.3.2 Track level quantities

The yields of tracks for a given p_T, η and φ are presented on figs. 3 to 6 for the $\sqrt{s} = 200$ GeV samples and on figs. 7 to 10 for the $\sqrt{s} = 510$ GeV samples. A good agreement was observed in both cases.

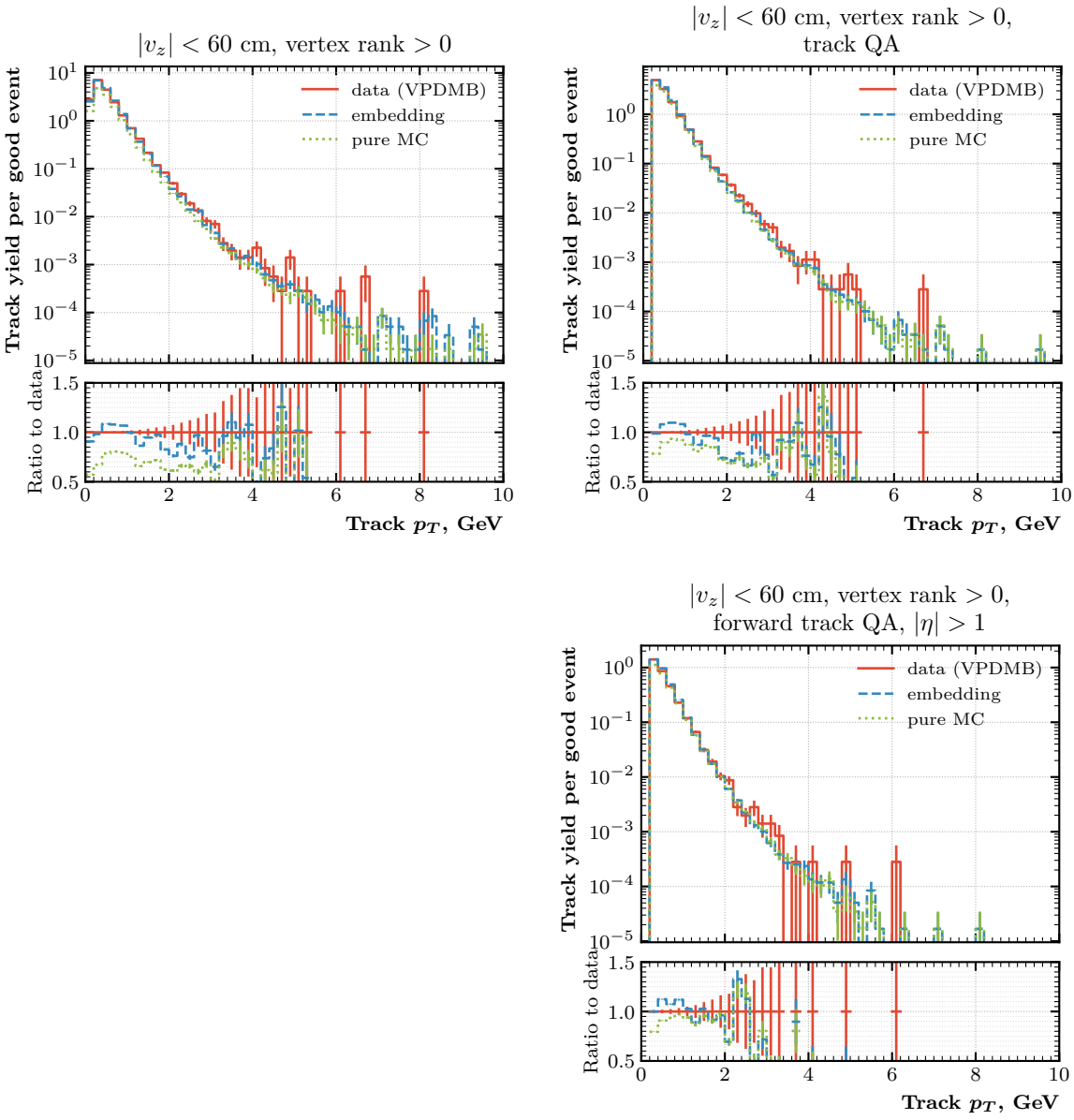

Figure 3: Track p_T distributions comparison between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

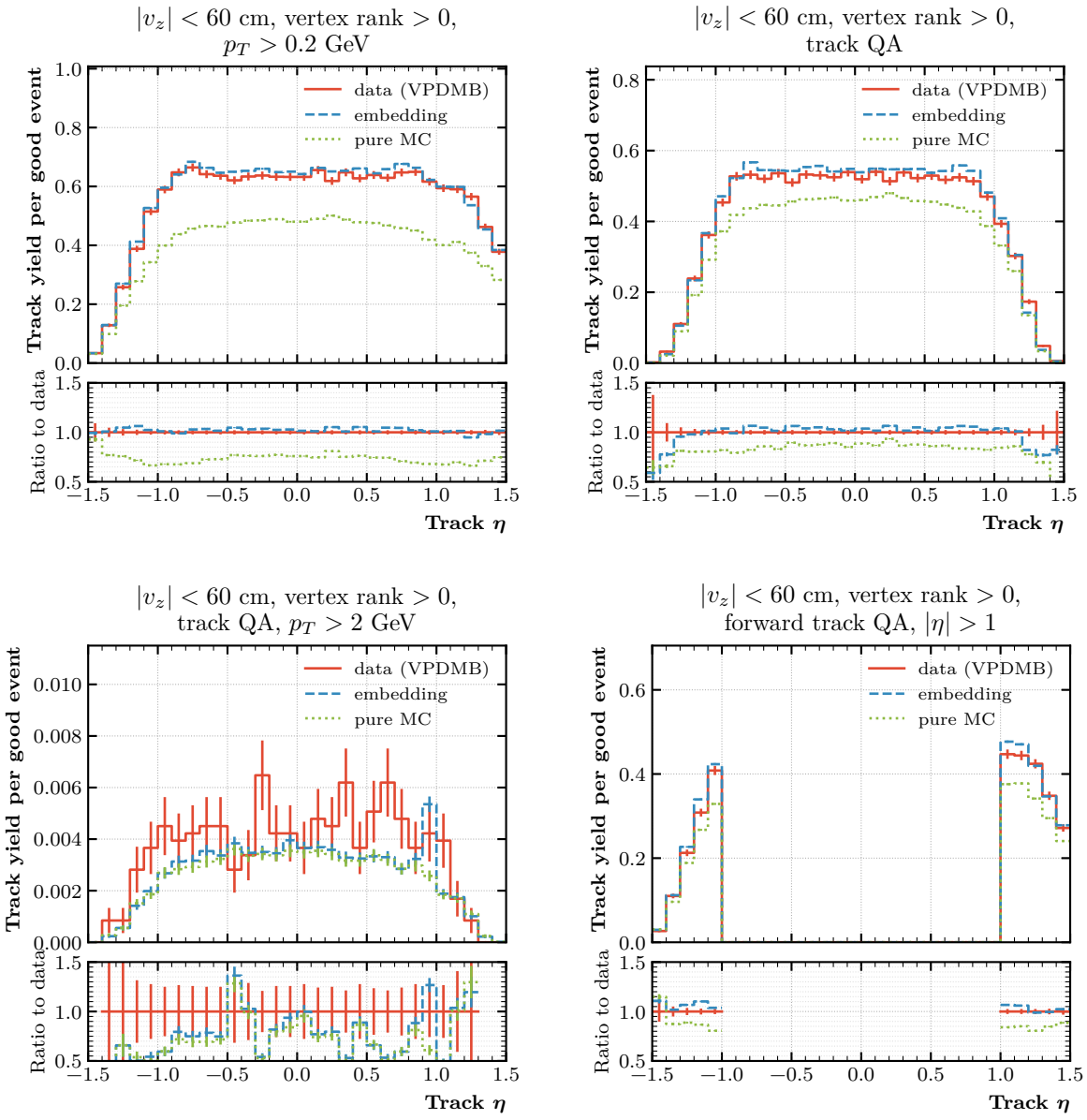

Figure 4: Track η distributions comparison between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

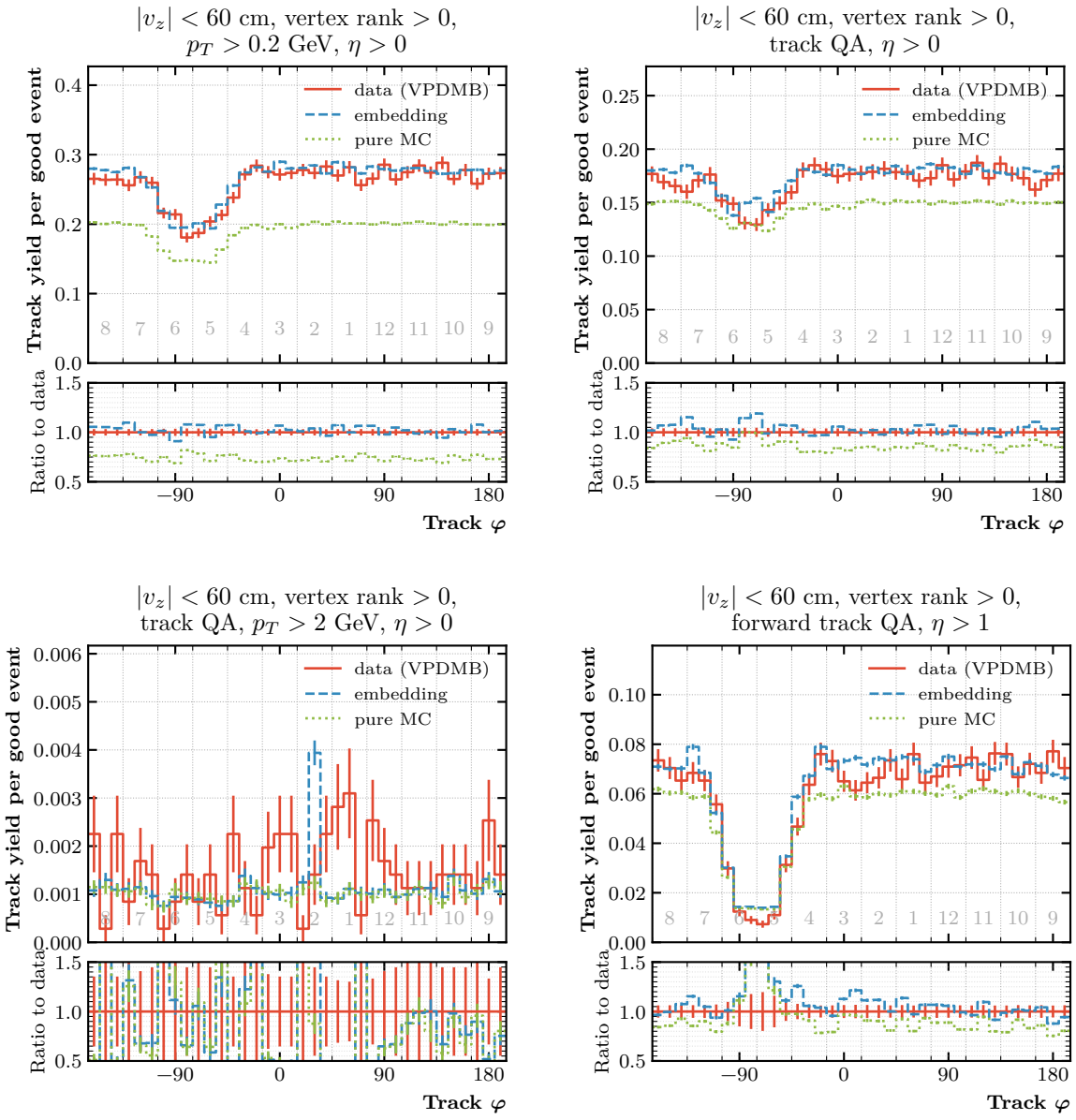

Figure 5: Track φ distributions comparison between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

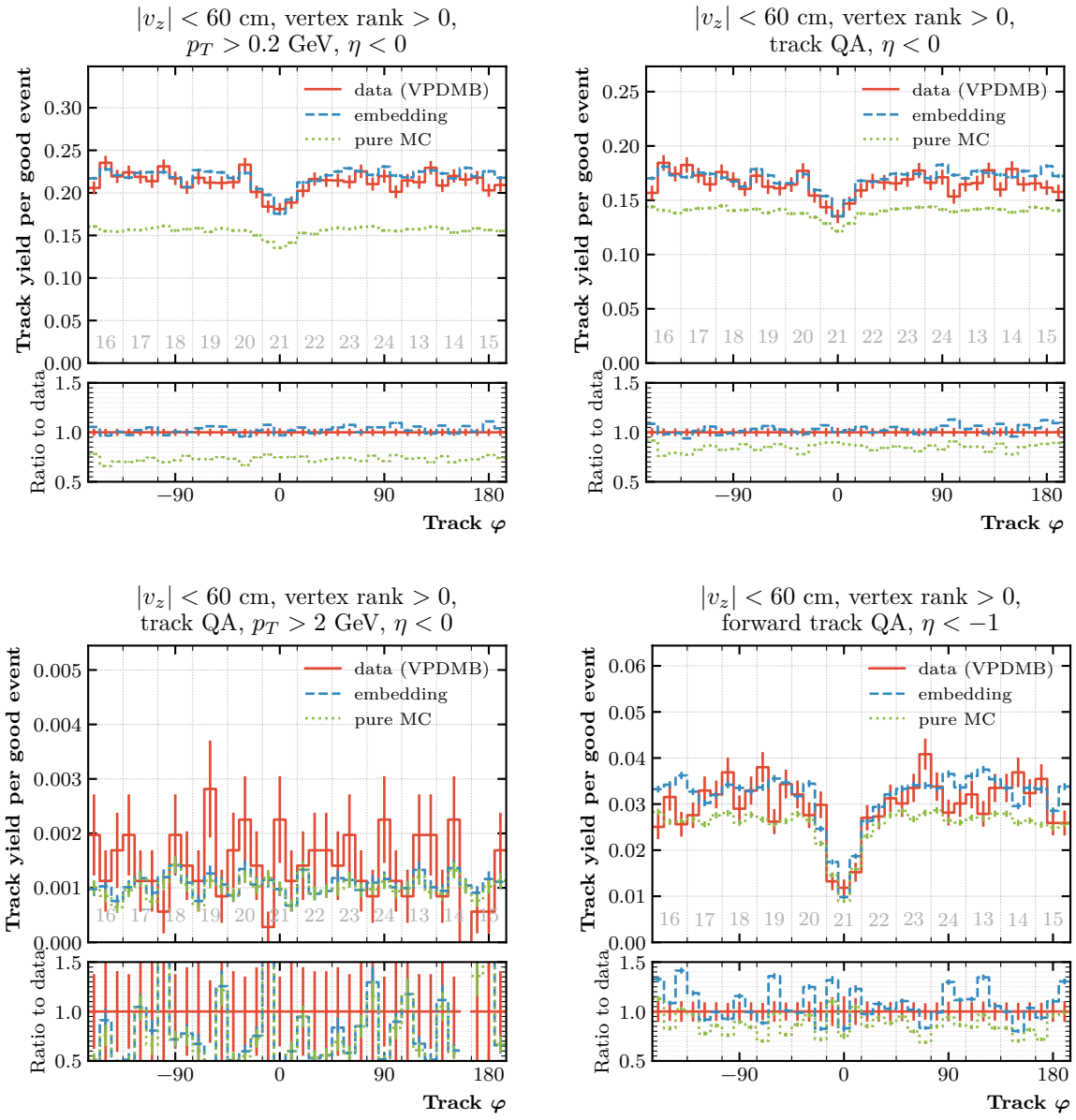

Figure 6: Track φ distributions comparison between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

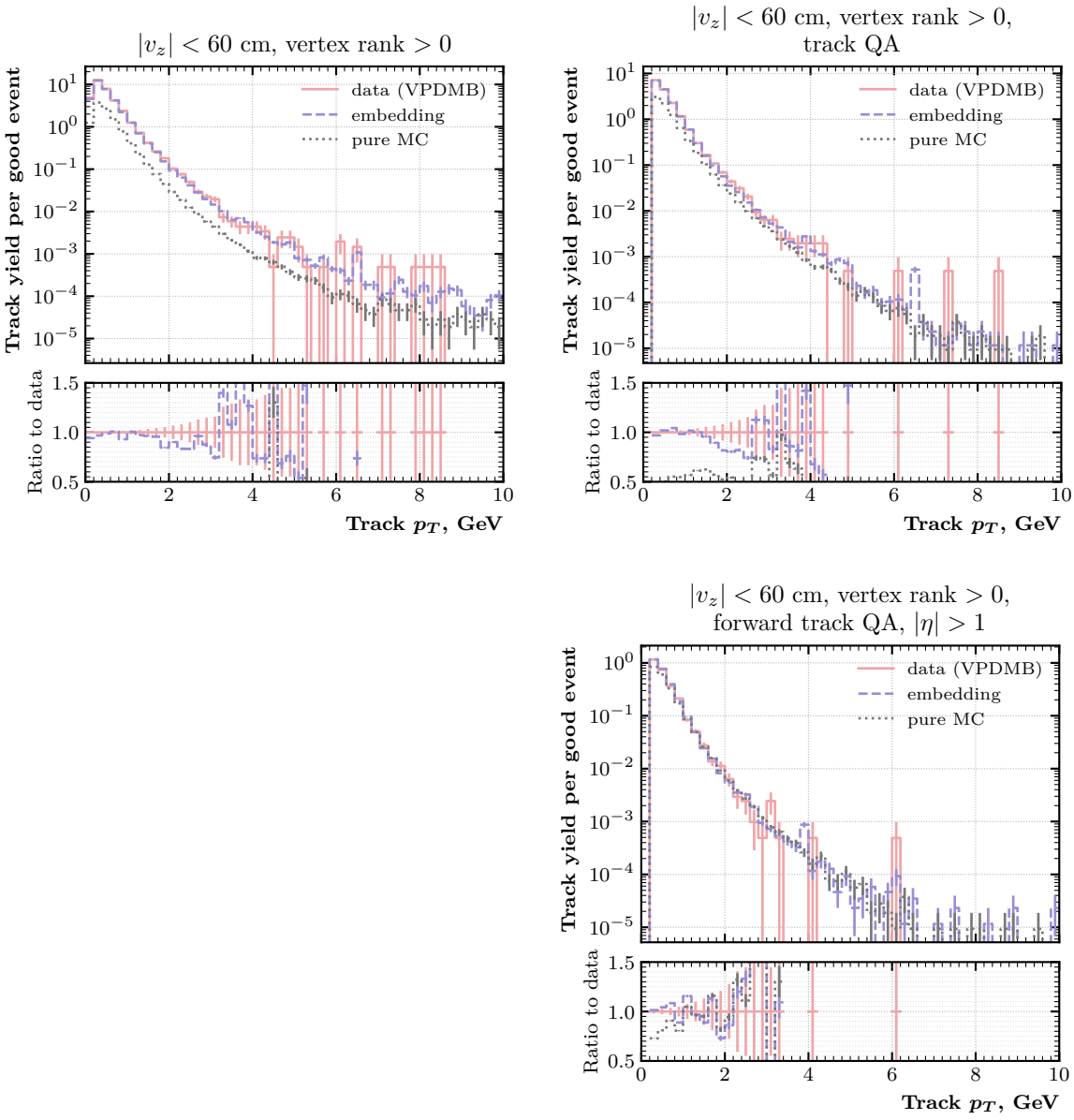

Figure 7: Track p_T distributions comparison between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

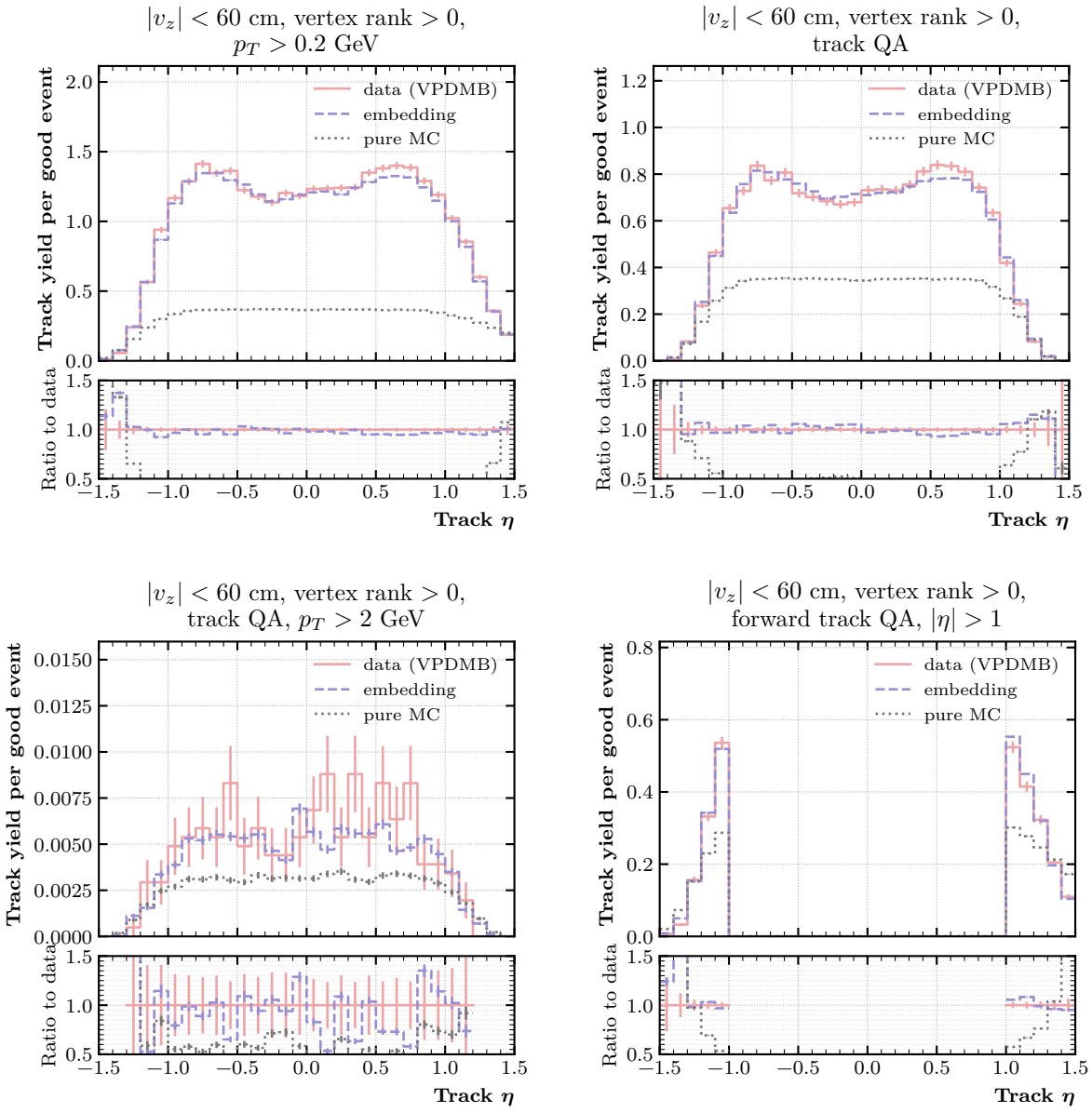

Figure 8: Track η distributions comparison between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

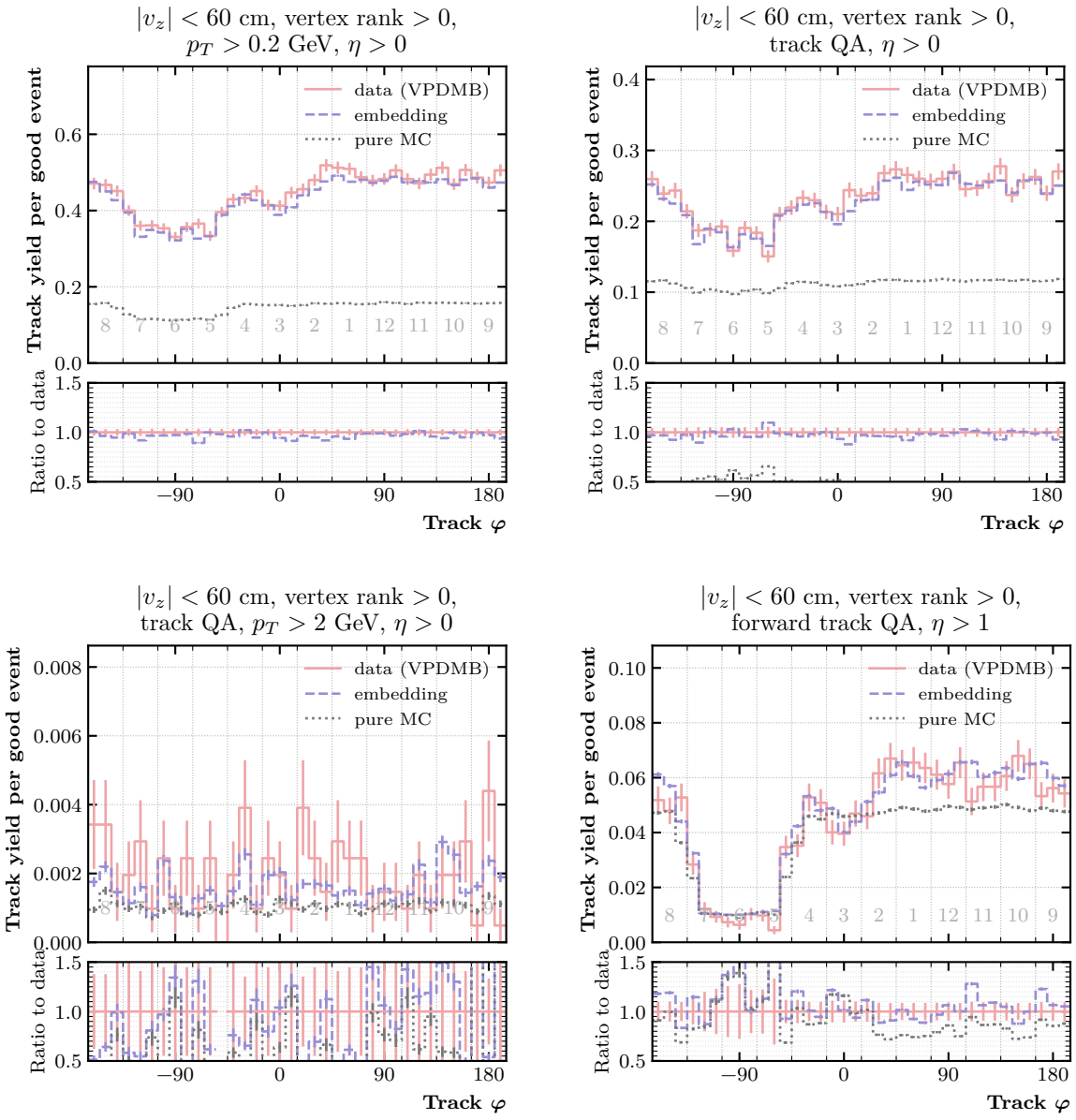

Figure 9: Track φ distributions comparison between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

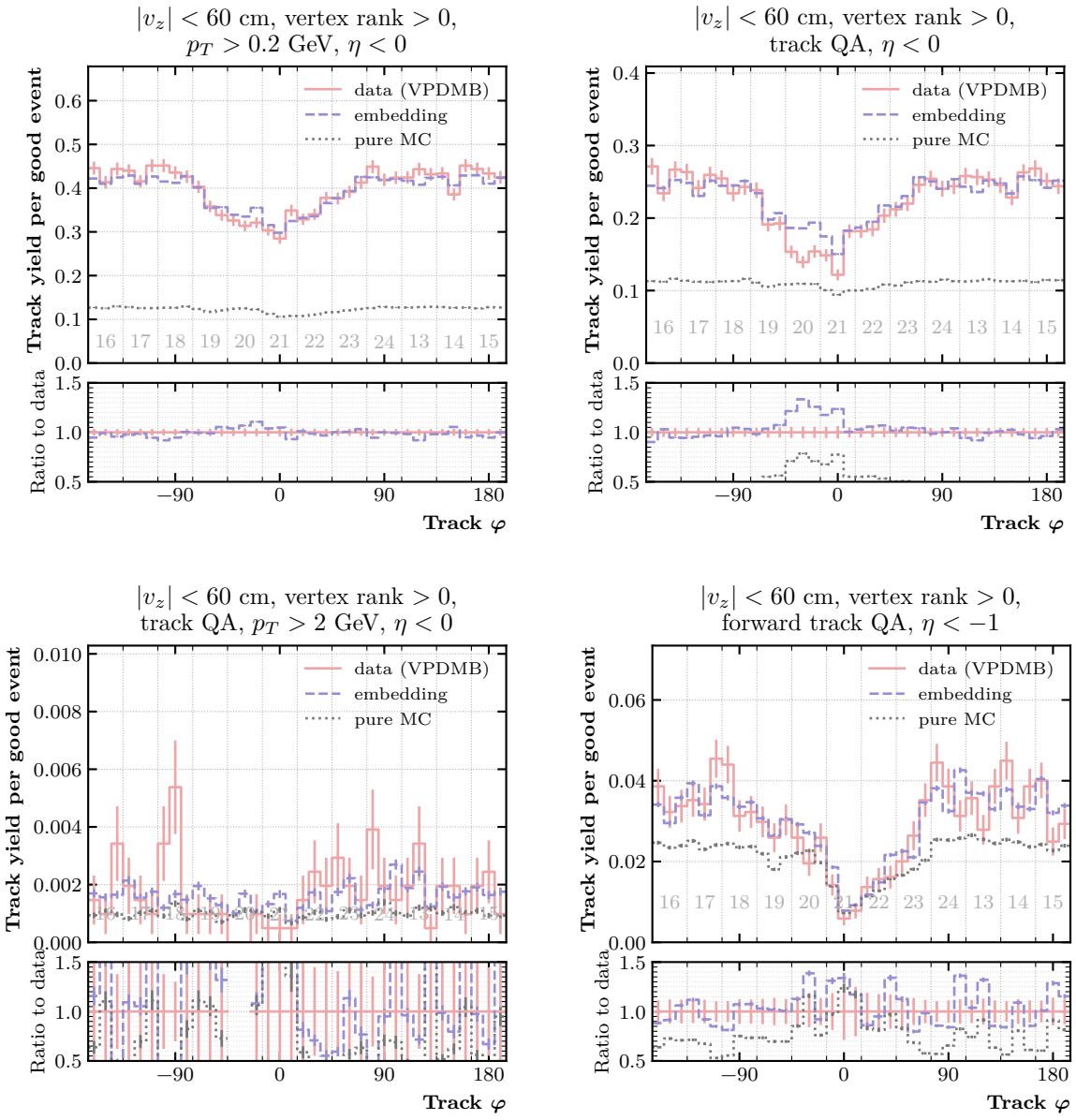

Figure 10: Track φ distributions comparison between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

5 Tracking efficiency for single embedded particles

In the past, the tracking efficiency uncertainty was estimated by first measuring the tracking efficiency for simulated tracks embedded into a data and into a simulation and then taking the difference between these results as an estimate for the uncertainty. Studies with the embedding of simulated tracks into data allows probing some but not all of the effects contributing to the tracking efficiency in the real detector. Namely, it probes how tracking performs in the context of a given event: including vertices and tracks present in it as well as any additional pile-up and detector noise contamination. We are looking to investigate if there are any such discrepancies that would affect the tracking efficiency. The study of the properties of the tracks measured in the data will be carried out in the section 6.

5.1 Tracking efficiency definition

Each event in our sample had a single particle embedded into it, so the tracking efficiency ϵ can be naturally estimated as a ratio of the numbers of events:

$$\epsilon = \frac{\#\text{events } \{\text{common cuts } \& \text{ single track reconstructed}\}}{\#\text{events } \{\text{common cuts}\}} \tag{1}$$

The "common cuts" are to select events that are representative of the events in the physics analyses. Here we require the following:

- the highest-ranking vertex has a positive rank
- $|v_z^{\text{HRV}}| < 60$ cm
- event has at least one "good track": a track that satisfies all of the track QA cuts (as defined in section 3.4) and is associated to the highest-ranking vertex. This good track should not match to the thrown particle.
- (optionally) a cut on the p_T, η, φ of the thrown particle – needed to study the tracking efficiency as a function of these variables
- (optionally) some additional event selection to match the physics analysis conditions (e.g. select the hard events)

The purpose of requiring a single "good track" is to reduce possible contribution of the vertex reconstruction efficiency. This condition is also enforced in a typical jet analysis via a cut on jet R_T.

The "single track reconstructed" condition requires all of the following:

- there is a primary track reconstructed in the event that is associated to the thrown particle (via `IdTruth`)
- that primary track is a "good track"

We consider track to be reconstructed only if it passes the physics analysis requirements (vertex association, track QA cuts) to provide a definition that is most applicable to our physics analyses. The association to the truth does not pass the tracks that were associated to the secondary particles (like muons from the pion decay) even though those may produce a reasonable TPC track.

5.2 Event selection

As described in section 4.1, the Pythia was used for generating the "unbiased" events. The VPDMB trigger condition was required for the data sample to approximately match that. We expect that such samples should be comparable, as it was previously shown that jet spectrum for the VPDMB trigger is only slightly softer than the unbiased trigger[9].

Two possibilities were explored for studying the hard events. The methods were chosen based on simplicity of their implementation. First was to select the Jet Patch triggers in the sample of the embedding into the real data. Second was to run anti-kT jet algorithm with $R = 0.6$ on the tracks to find charged jets in the event and then require charged jets of a certain p_T to be present in the event. No statistically significant deviations were found in both of those studies.

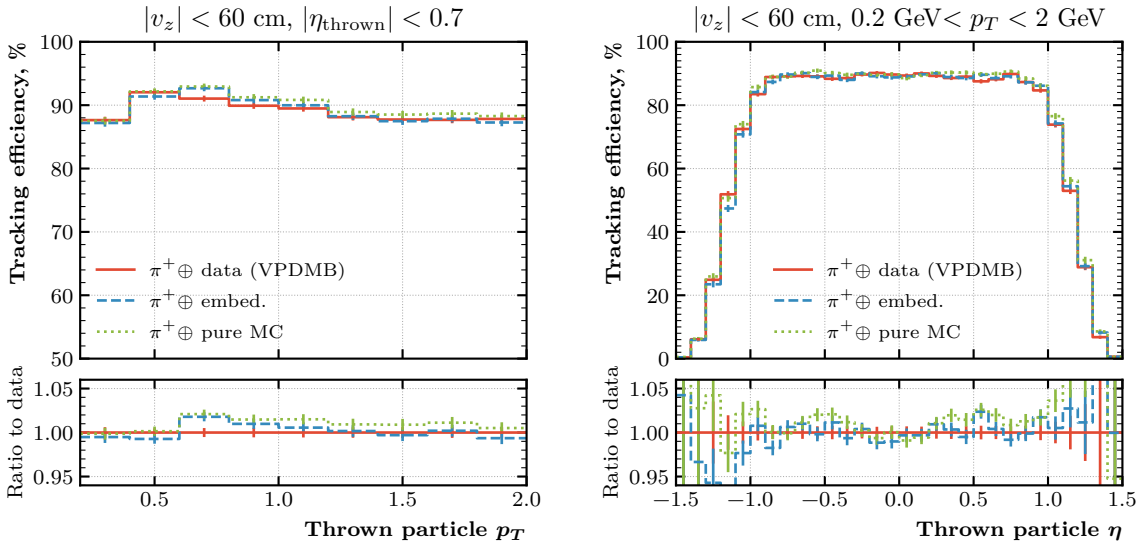

Figure 11: Tracking efficiency as a function of track p_T and track η in run 13059025 ($\sqrt{s} = 200$ GeV).

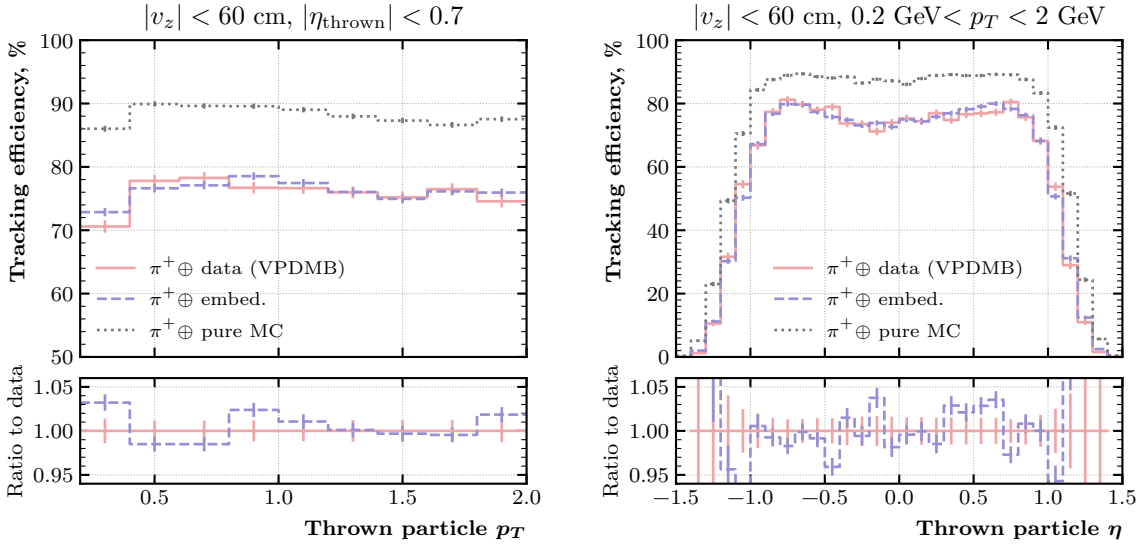

Figure 12: Tracking efficiency as a function of track p_T and track η in run 13104003 ($\sqrt{s} = 510$ GeV).

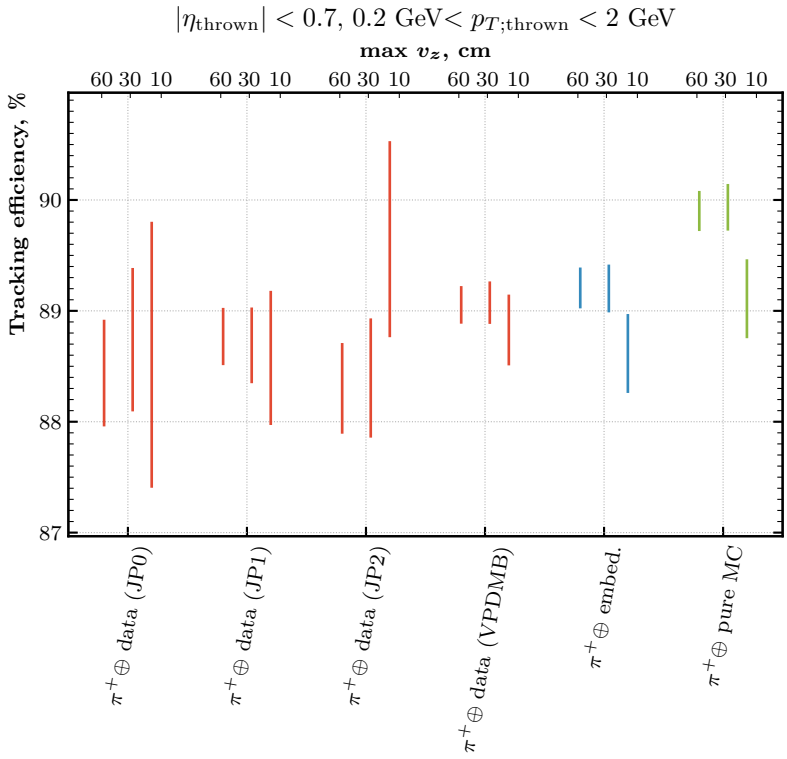

Figure 13: Tracking efficiency in the mid-rapidity region for various values of vertex z cut for run 13059025 ($\sqrt{s} = 200$ GeV).

5.3 Results

The efficiency as a function of p_T and η is presented on figs. 11 and 12. We see a good agreement between data and embedding in both of these samples. The minor discrepancies at high values of $|\eta|$ are expected to be caused by small differences in the vertex distributions. At $\sqrt{s} = 200$ GeV (fig. 11) we observe that the efficiency for pure MC sample is only slightly higher the embedding, which implies that pile-up produced only a small effect at this energy. It is, however, produces a bigger effect at $\sqrt{s} = 510$ GeV (fig. 12) where a significant reduction in efficiency was observed with absolute differences ranging from 8% to 14% depending on the value of pseudorapidity η.

For the purposes of providing a single number, the tracking efficiency was averaged for the embedded particles with $|\eta| < 0.7$ and $p_T \sim \text{Uniform}(0.2\text{ GeV}, 2\text{GeV})$[4]. The results for both center-of-mass energies are shown on figs. 13 and 14. The possible sensitivity to the hardness of the event was studied by requiring the Jet Patch trigger in the data, no significant effect was observed. The cut on the maximal absolute value of vertex z position was varied from 60 cm to 30 cm and to 10 cm, which also had no significant effect on the average tracking efficiency in the mid-rapidity region.

We see that, overall, the tracking efficiencies for simulated tracks embedded into the data and for simulated tracks embedded into the embedding agrees with a subpercent precision.

5.4 Cut pass rates

The tracking efficiency can be decomposed into individual components such as an efficiency of reconstruction for the global tracks, an efficiency of association to the vertex and an efficiency of passing the quality cuts. We will refer to the latter as "cut pass rate". It was estimated as a following quantity:

$$\epsilon = \frac{\#\text{tracks } \{\text{cut of interest } \& \text{ good true track}\}}{\#\text{tracks } \{\text{good true track}\}} \tag{2}$$

The "good true track" condition requires all of the following:

[4]This was also studied in another sample where the p_T of the embedded particle was fixed at 0.3 GeV, the final numbers were rather close to those included in this note.

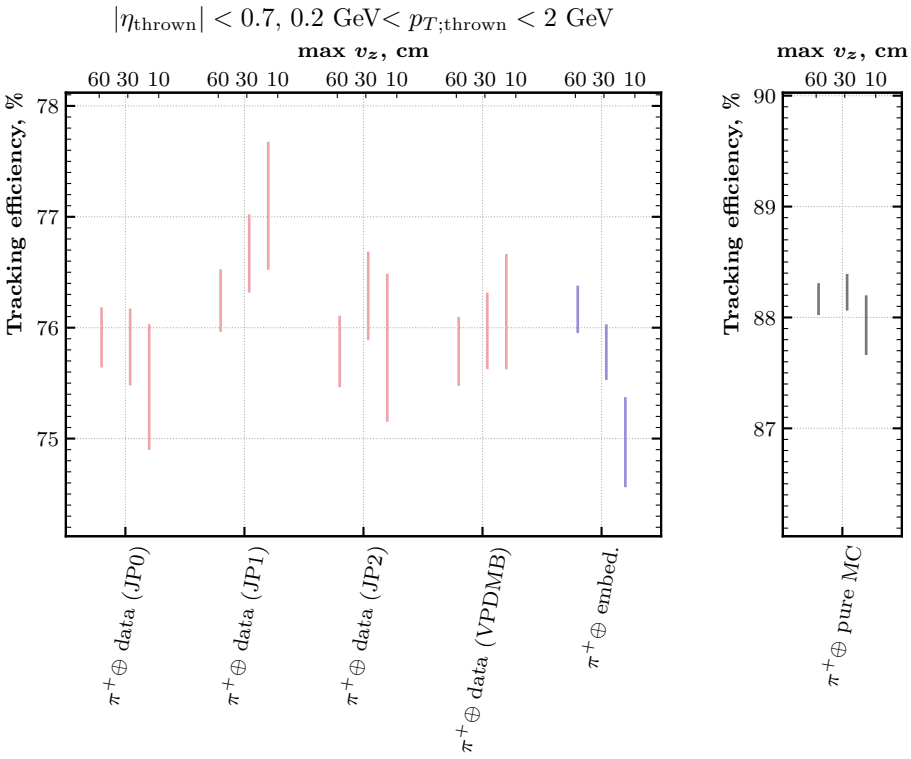

Figure 14: Tracking efficiency in the mid-rapidity region for various values of vertex z cut for run 13104003 ($\sqrt{s} = 510$ GeV).

- the track is associated to the highest-ranking vertex that has a positive rank and $|v_z^{\text{HRV}}| < 60$ cm
- the track is associated to the thrown particle (via `IdTruth`)
- a cut on the p_T, η, φ of the reconstructed track – needed to study the cut pass rate as a function of these variables

The "cut of interest" is one of the track QA cuts as described in section 3.4.

The results for the "cut pass rate" are shown on figs. 15 and 16. The subpercent agreement is confirmed for the $\sqrt{s} = 200$ GeV, whereas for the $\sqrt{s} = 510$ GeV the agreement is only seen within the available statistics.

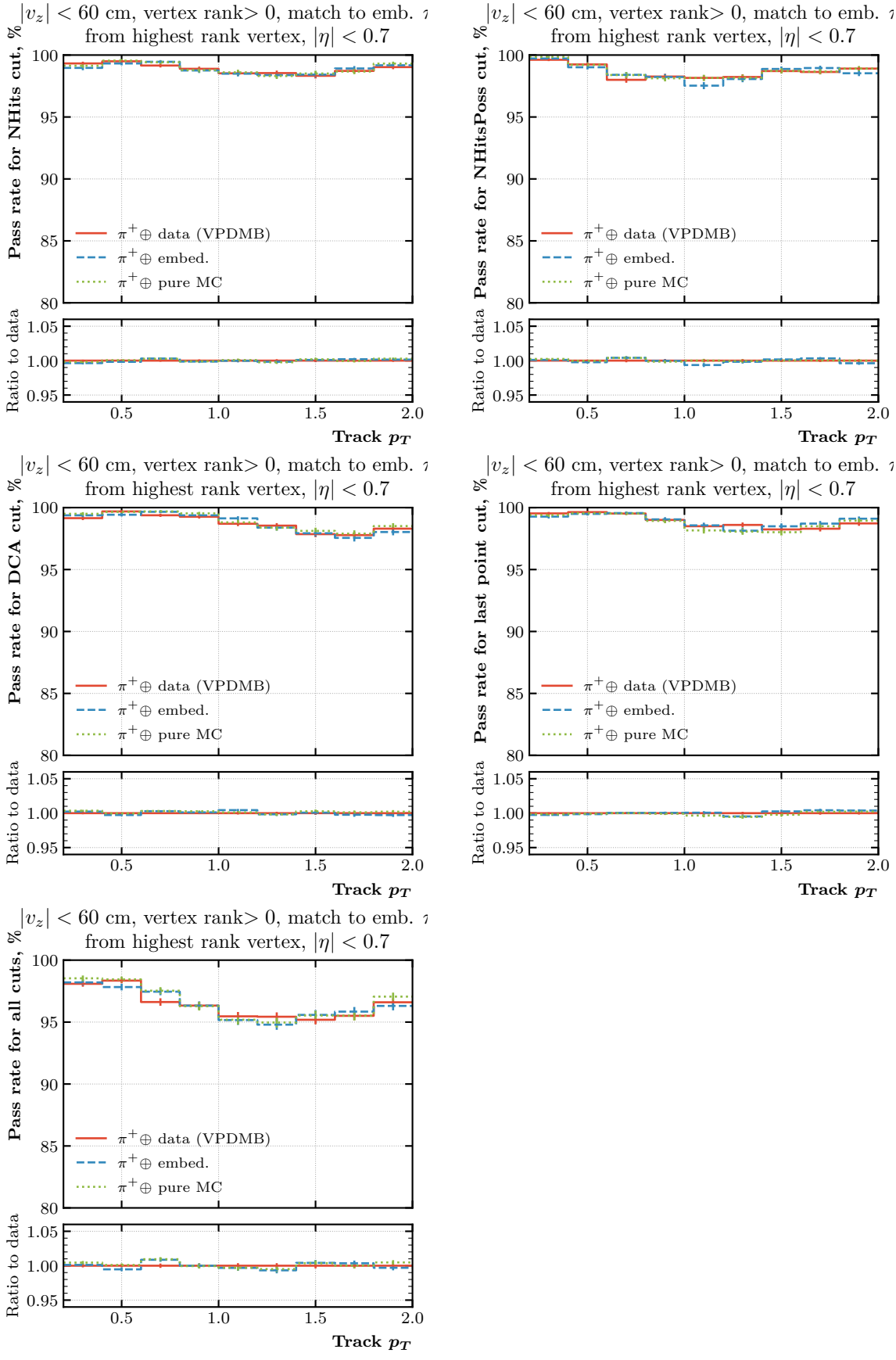

Figure 15: Fraction of tracks associated to the thrown particle that pass a given track QA cut in run 13059025 ($\sqrt{s} = 200$ GeV).

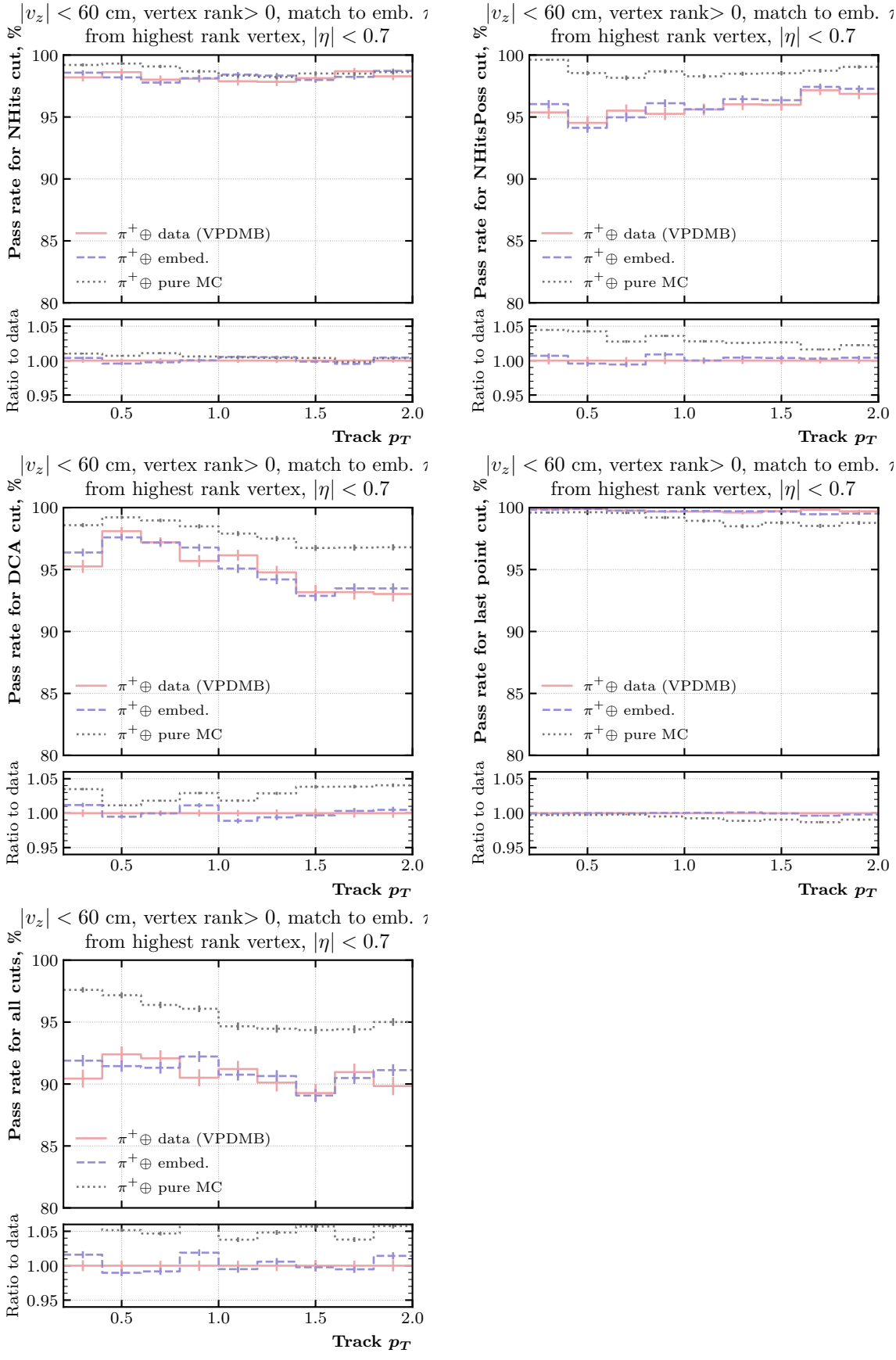

Figure 16: Fraction of tracks associated to the thrown particle that pass a given track QA cut in run 13104003 ($\sqrt{s} = 510$ GeV).

6 Comparison of the track properties

The tracking efficiency study using simulated tracks does not take into account the possibility that the tracks in the data might have different properties than the simulated tracks. The discrepancies will occur depending on how inaccurately the simulation reproduces materials that particle passes before entering the volume inside the TPC or after in its interaction with the TPC gas. It should be also important that the charge drift, avalanche and digitization are also correctly simulated. Additionally, any inadequancy in the correction for distortions in the data may be introducing effects that are not reproduced in the embedding. We believe that such differences, if present, should manifest themselves as discrepancies in the distributions of the number of hits on the tracks or in distributions of the DCA of the tracks. The discrepancies in these quantities will affect the fraction of the tracks that pass the related track selection criteria and subsequently contribute a discrepancy to the effective tracking efficiency.

6.1 Track yields

An excellent agreement was observed in the number of hits distributions in figs. 17 and 20. A more detailed comparison for hits in the inner and outer TPC regions is shown on figs. 18, 19, 21 and 22. Some significant discrepancies were seen when looking at the DCA distributions (shown on figs. 24 and 29 and on figs. 25 to 28 and 30 to 33 as averages vs the track p_T, η and ϕ). An effect of an improper space charge correction in the data should be visible in the signed DCA distributions (shown on figs. 34 and 39 and on figs. 35 to 38 and 40 to 43 as averages vs the track p_T, η and ϕ). The DCA resolution is better in simulation that it is in the data. A method for smearing DCA in the simulation is described in [10].

6.2 Single hit efficiency

An attempt at measuring the hit efficiency was made by studying tracks that 44 or 45 hits. Tracks with 44 hits have a single hit missing. The "topology map" was inspected to determine the position of the missing hit and a yield of the 44-hit tracks was calculated versus the missing hit position, these yields were then normalized by the total number of the 45-hit tracks. This quantity shown on fig. 23. Because the distributions of number of hits (figs. 18 and 19) do not behave like a binomial distributions we can not simply relate the presented quantity to the real hit efficiency, we do, however, believe that it should have a similar magnitude as the hit efficiency in a given TPC padrow. Thus, the result indicates a low hit *in*-efficiency, which, given a requirement of at least 12 hits on the track, can not make a substantial contribution to the to the tracking *in*-efficiency.

6.3 Cut pass rates

The differences between the DCA distributions for the data and embedding indicate a possibility for a difference in the tracking efficiencies. One possible way to quantify the uncertainty would be to estimate cut pass rates in a way similar to how it was done in section 5.4, but, in this case, for all tracks in the events.

The cut pass rates for all tracks versus the track p_T is shown on figs. 44 and 48. As expected, the result for the cut pass rate shows an excellent agreement for all cuts of interest except for the DCA cut. In the case of $\sqrt{s} = 200$ GeV, we see an agreement within $\sim 1\%$ and for $\sqrt{s} = 510$ GeV it is on the order of $\sim 2\%$.

When looking at the cut pass rate for the DCA cut, one needs to remember that those relative to the intrinsic 3 cm cutoff required for tracks at reconstruction. The uncertainty associated with the DCA cut can vary from a low uncertainty at high background for a wide cut to a higher uncertainty at lower background for a tighter cuts. The size of background in embedding can be approximated by the difference between embedding and pure MC curves on fig. 24 for $\sqrt{s} = 200$ GeV and fig. 29 for $\sqrt{s} = 510$ GeV.

Looking at the specific region of tracks with $p_T > 0.5$ GeV and $-0.7 < \eta < 0$ on figs. 52 and 53 we see a specific feature in the pp510 data sample at $\varphi \simeq -30°$, which corresponds to the sector 20 of the TPC [11].

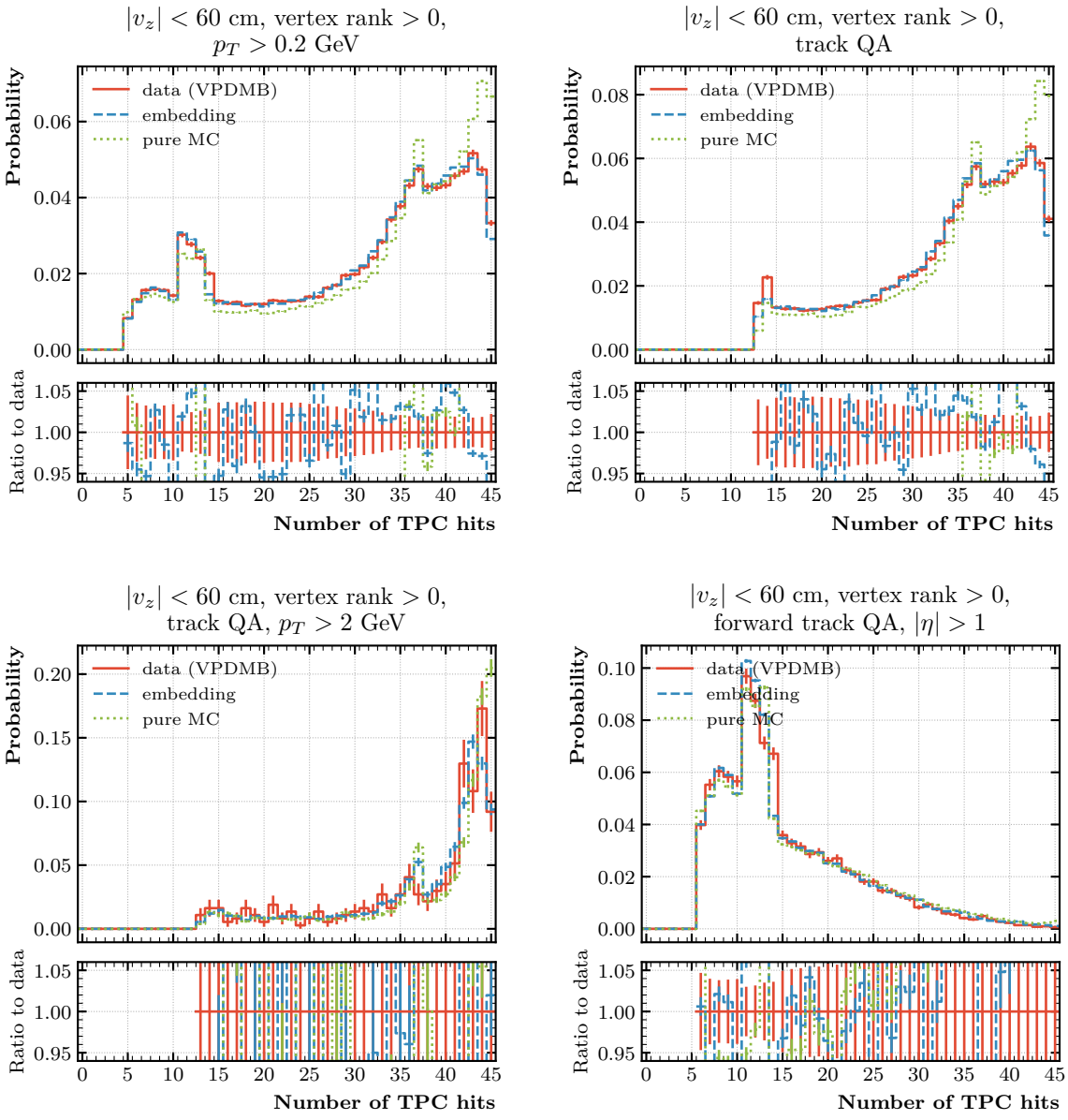

Figure 17: Distributions of the number of TPC hits per track compared between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

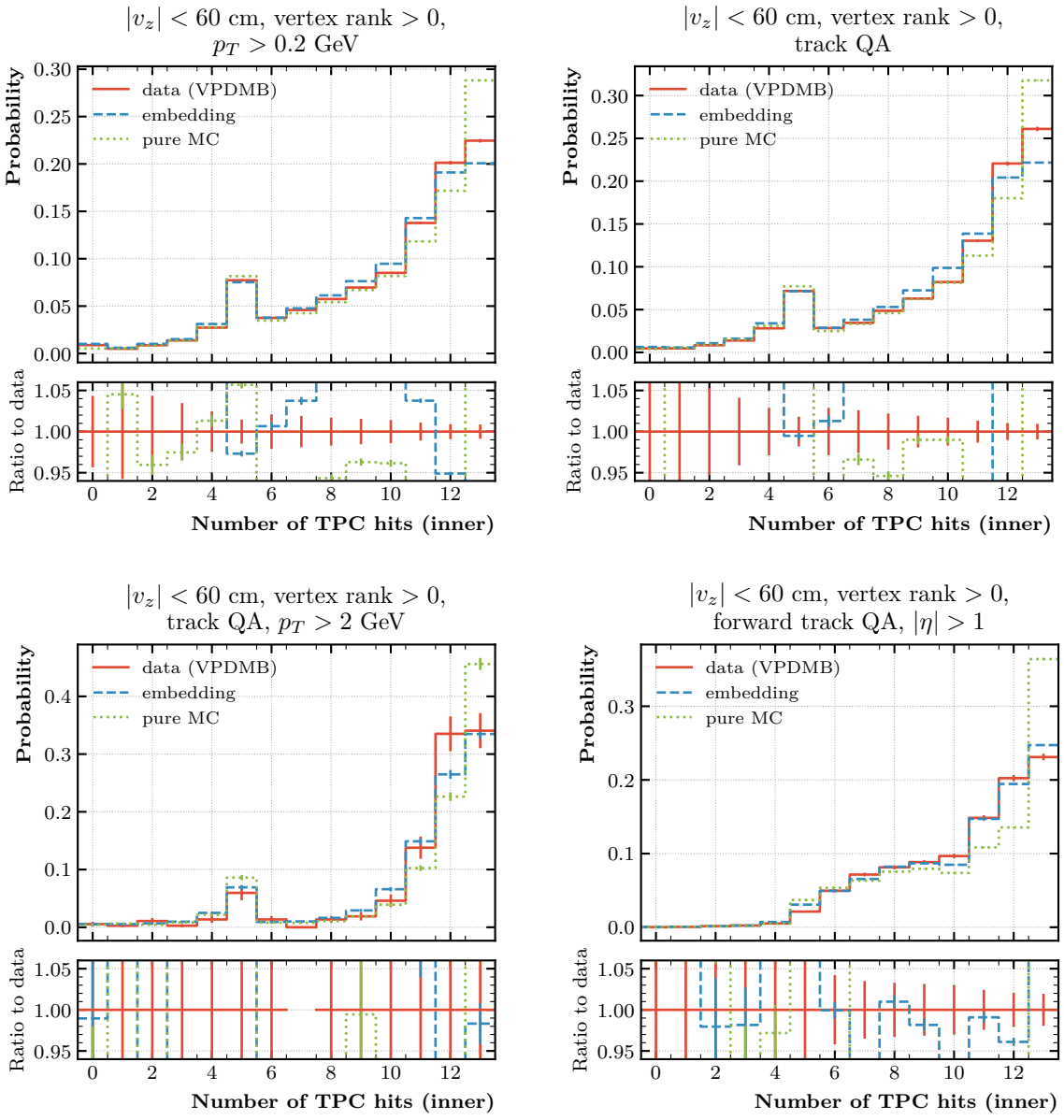

Figure 18: Distributions of the number of inner TPC hits per track compared between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

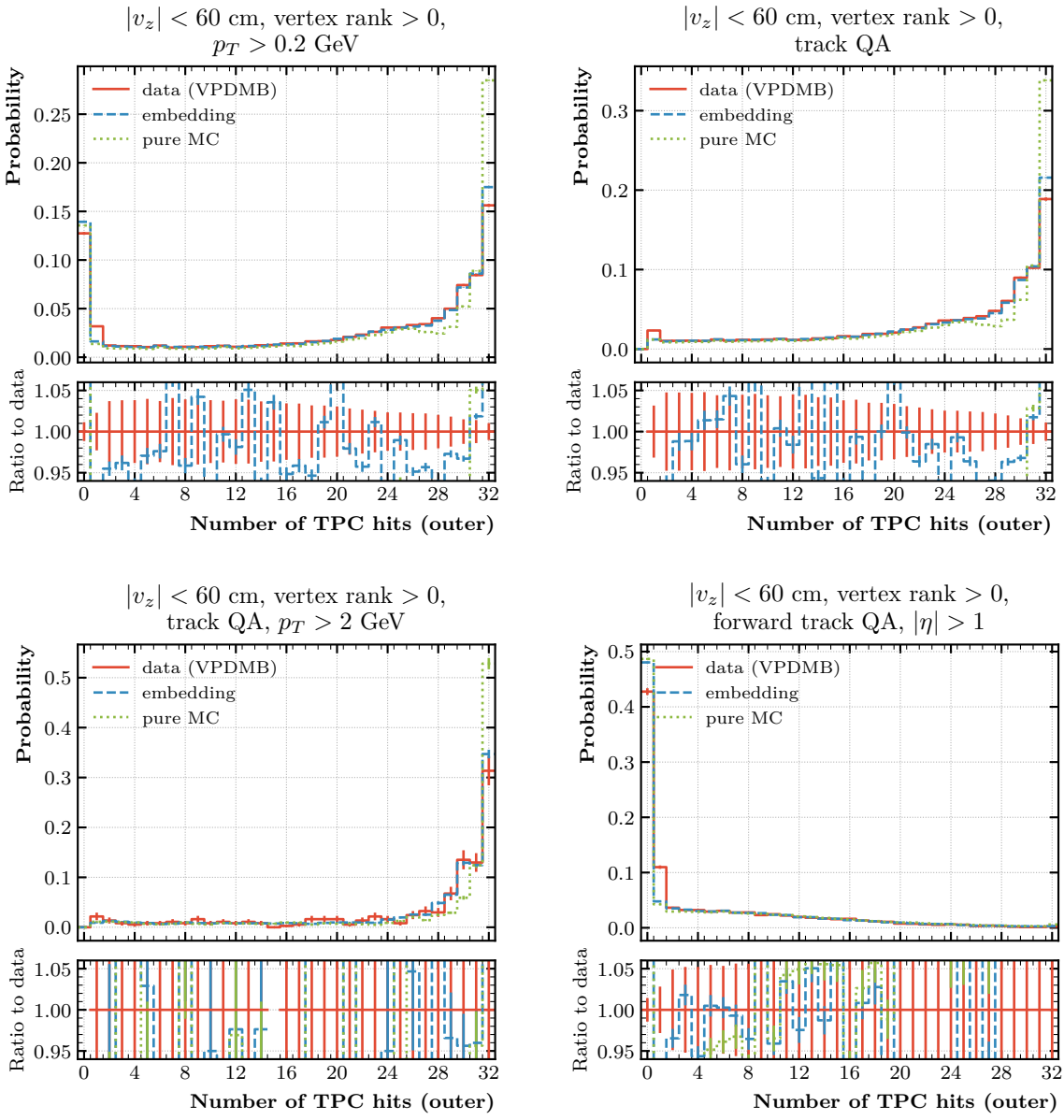

Figure 19: Distributions of the number of outer TPC hits per track compared between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

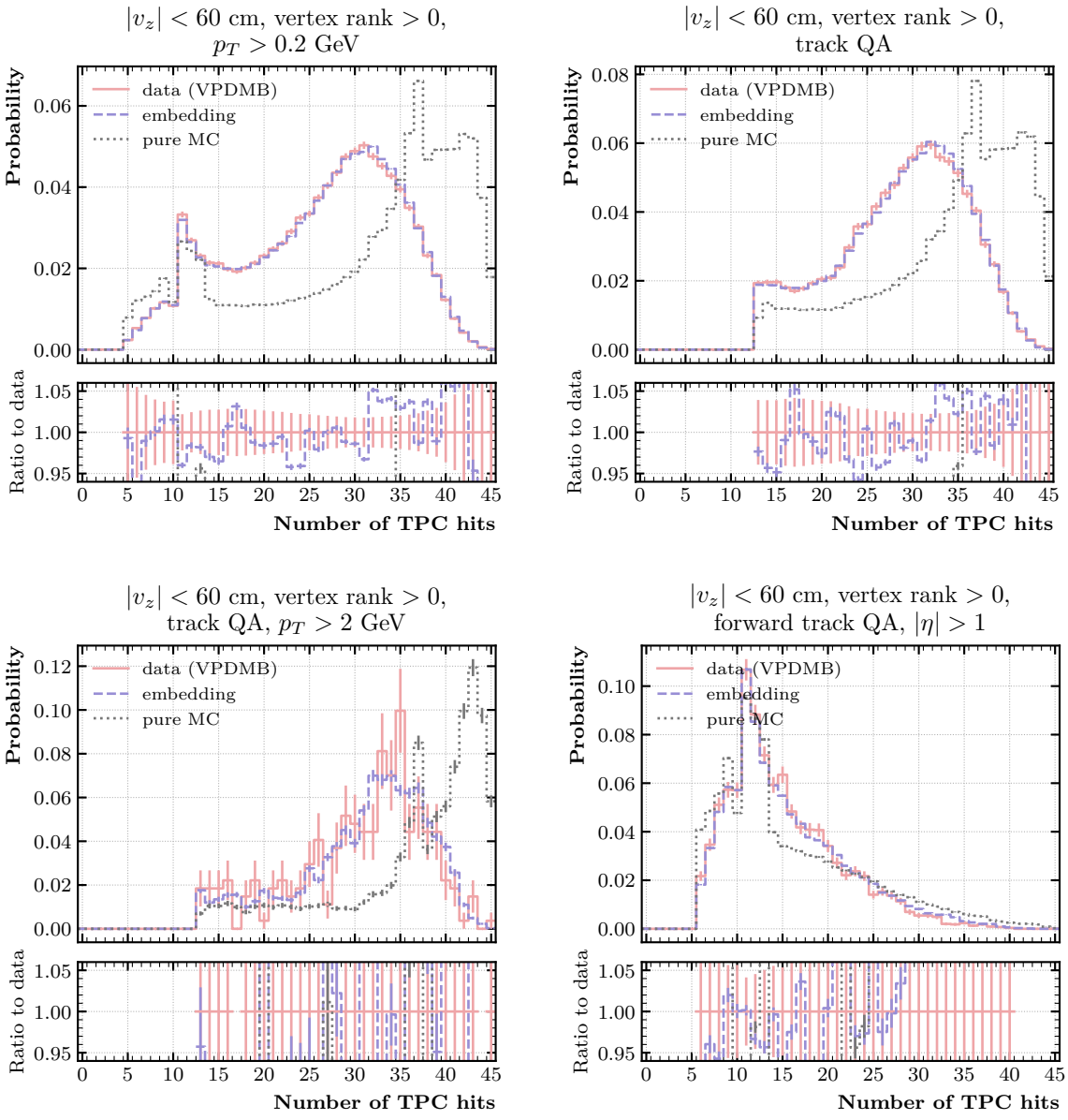

Figure 20: Distributions of the number of TPC hits per track compared between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

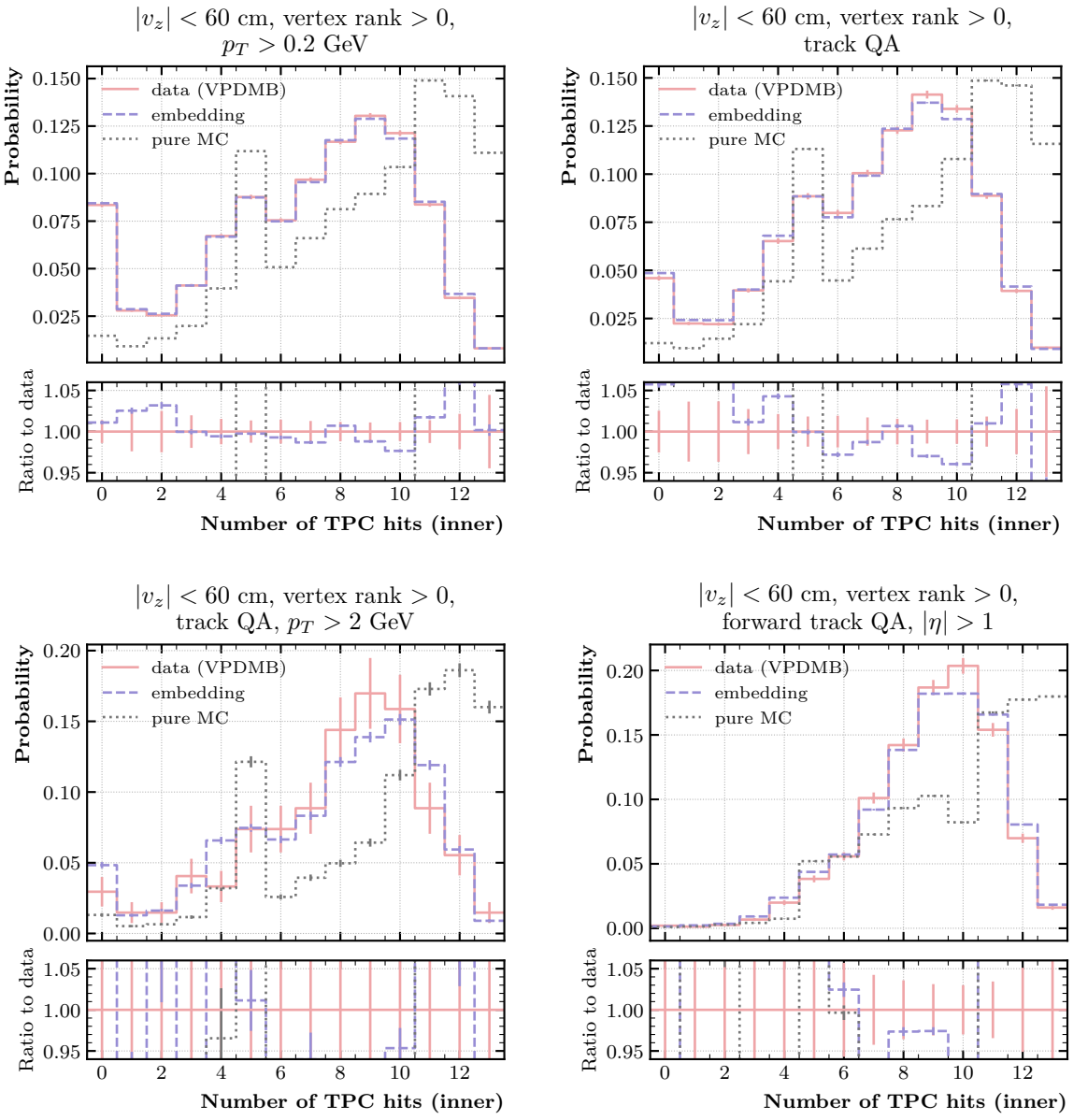

Figure 21: Distributions of the number of inner TPC hits per track compared between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

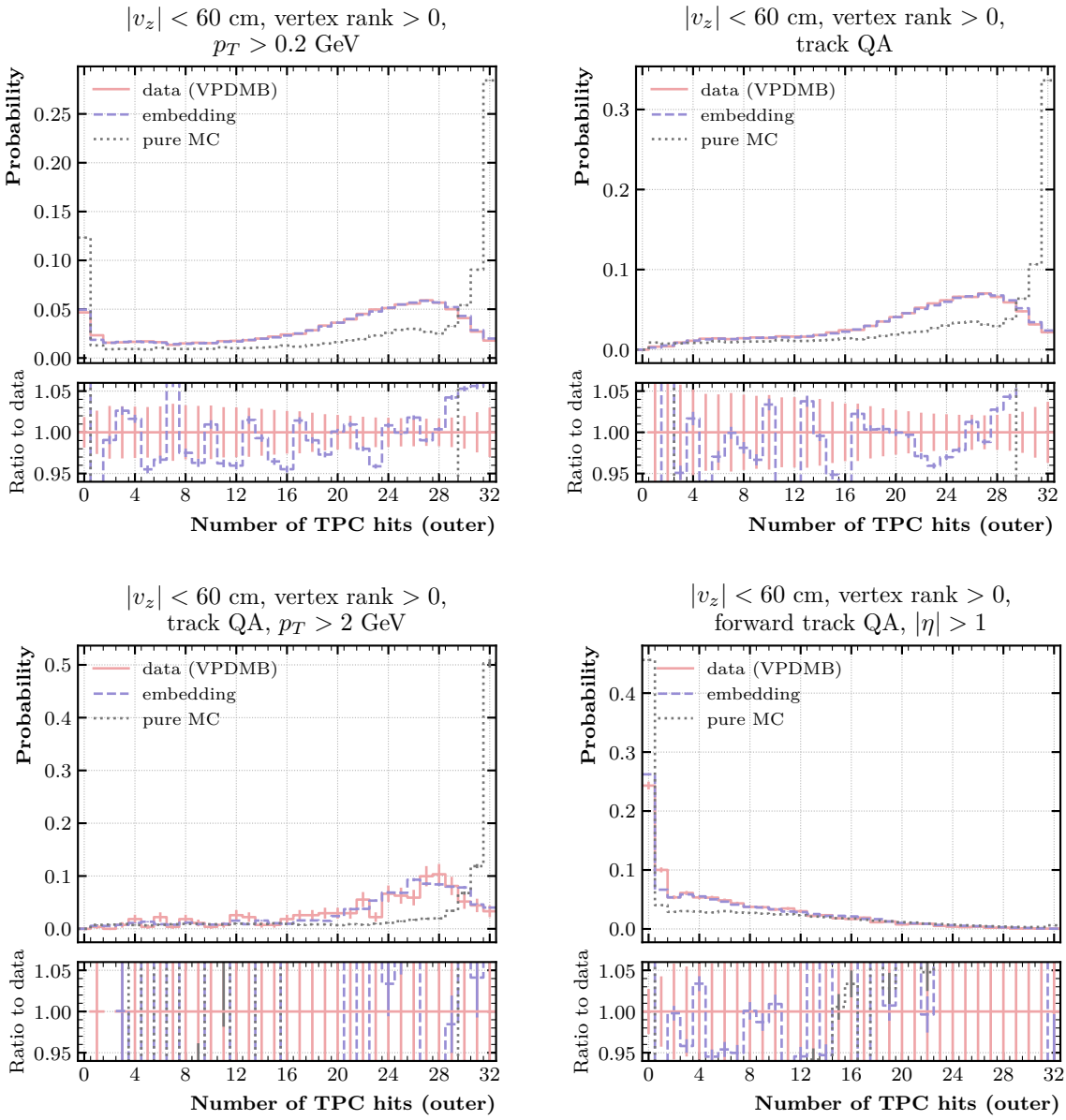

Figure 22: Distributions of the number of outer TPC hits per track compared between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

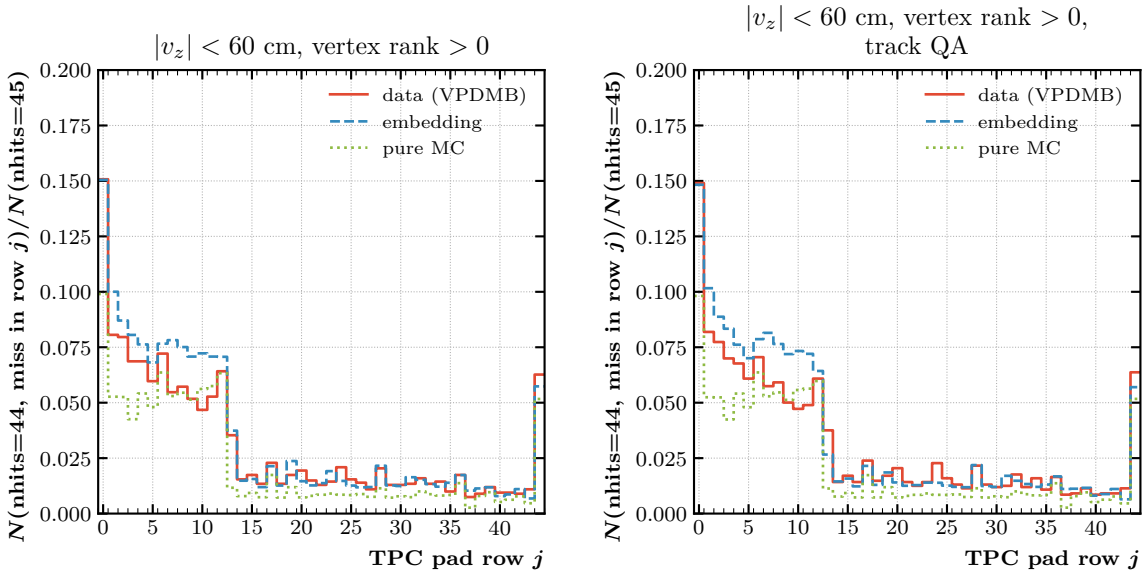

Figure 23: Distributions of the rate for missing a single hit in certain layer compared between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV).

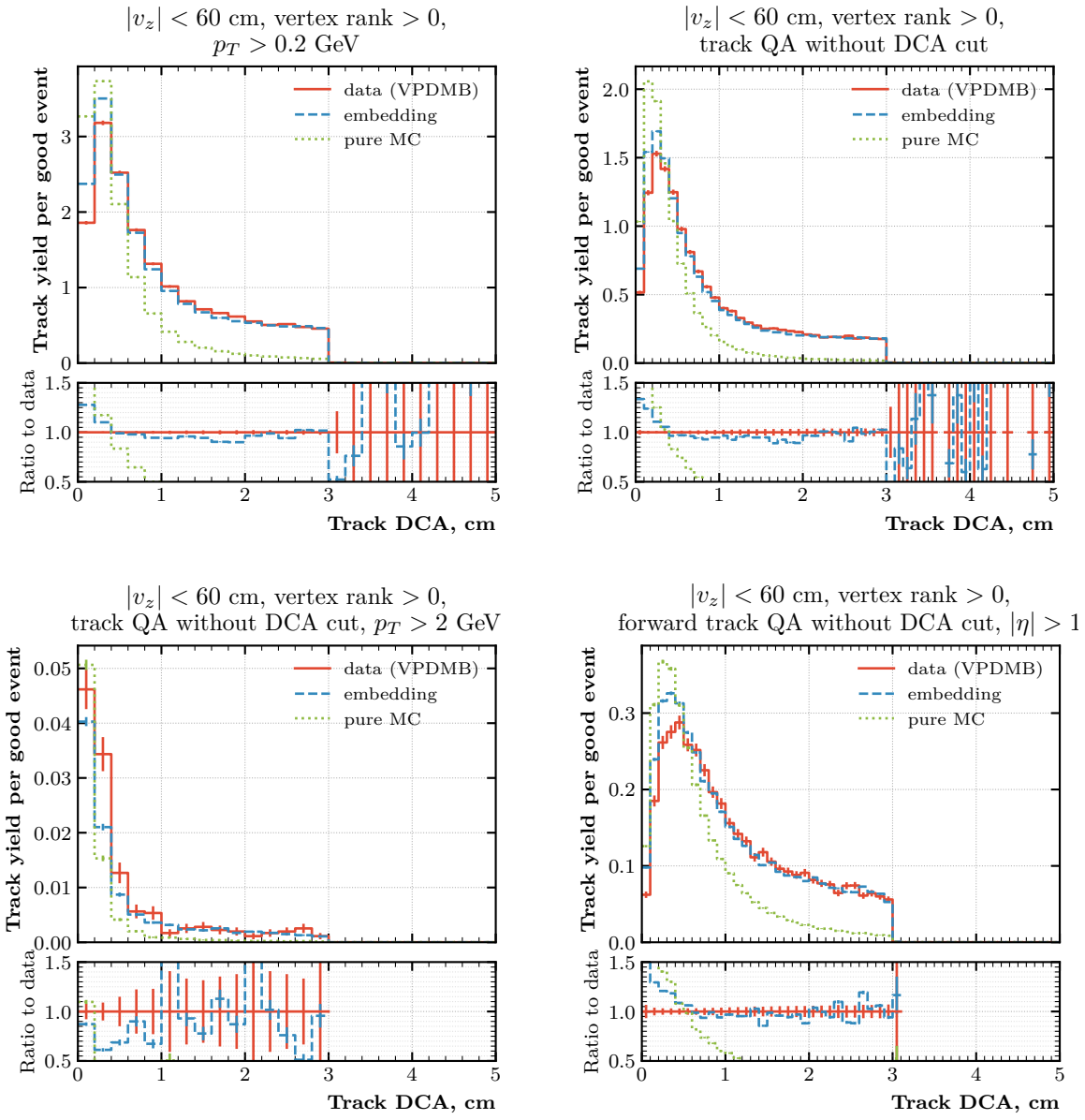

Figure 24: Track DCA distributions comparison between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

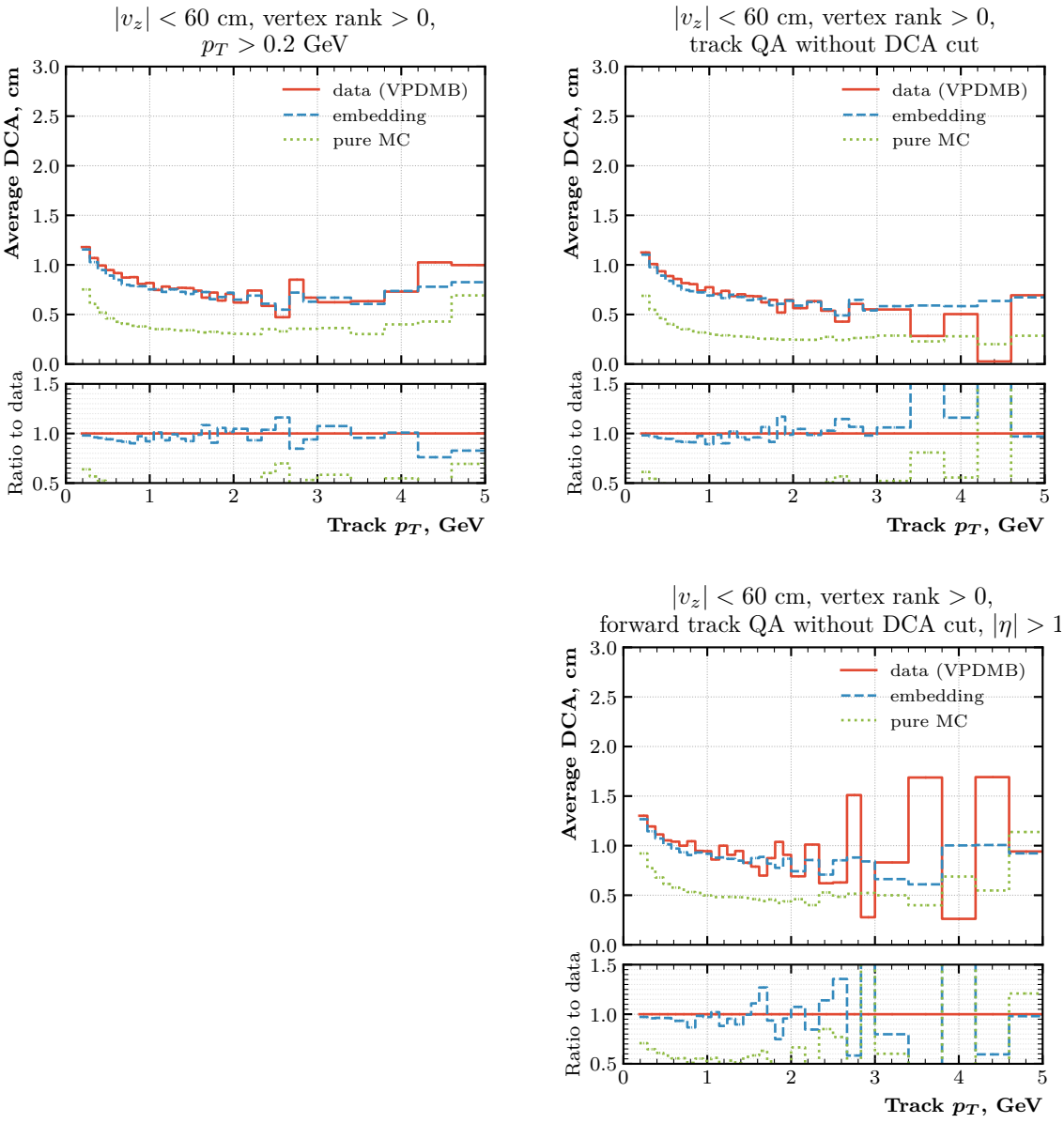

Figure 25: Average track DCA for different track p_T compared between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

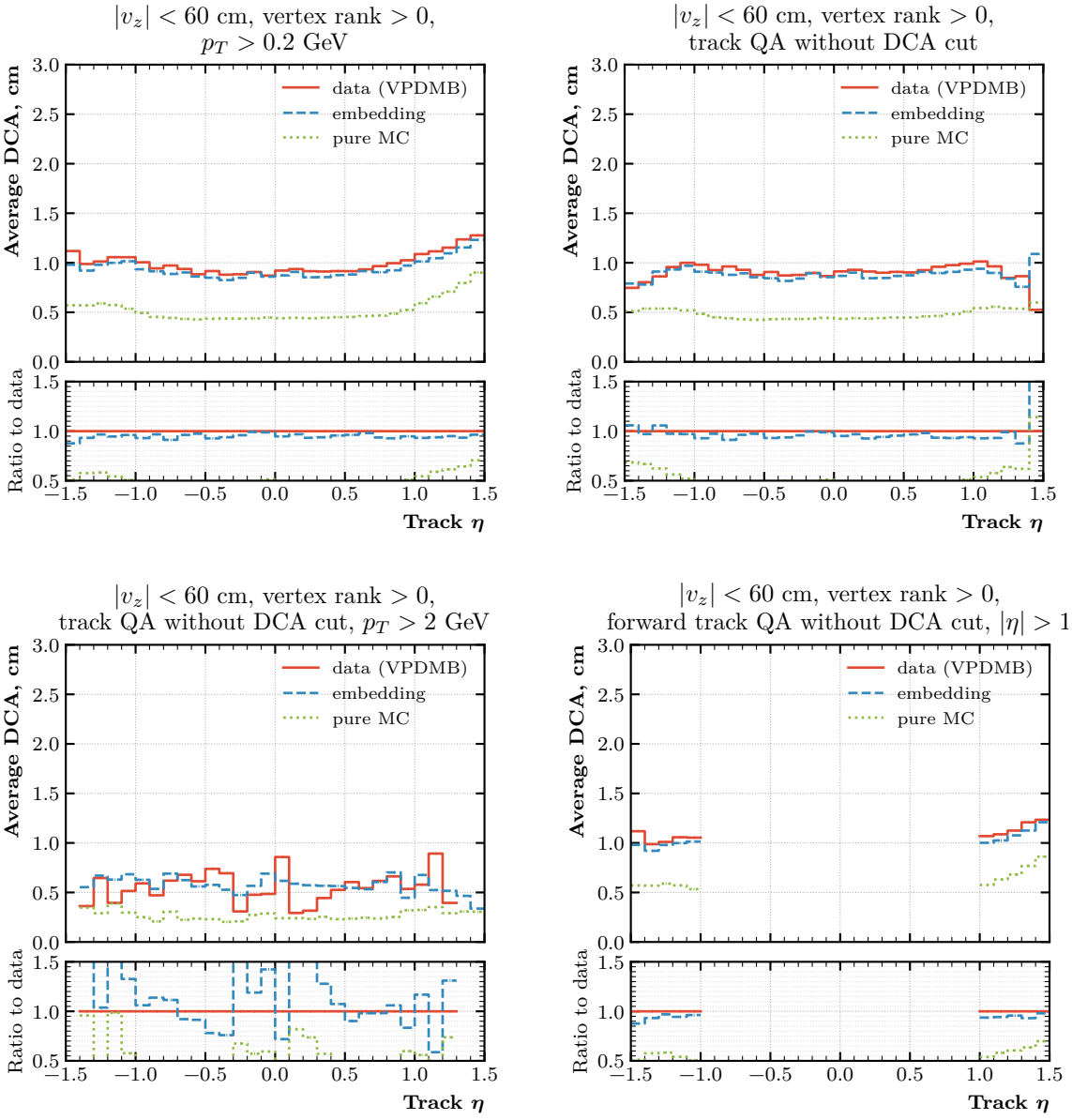

Figure 26: Average track DCA for different track η compared between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

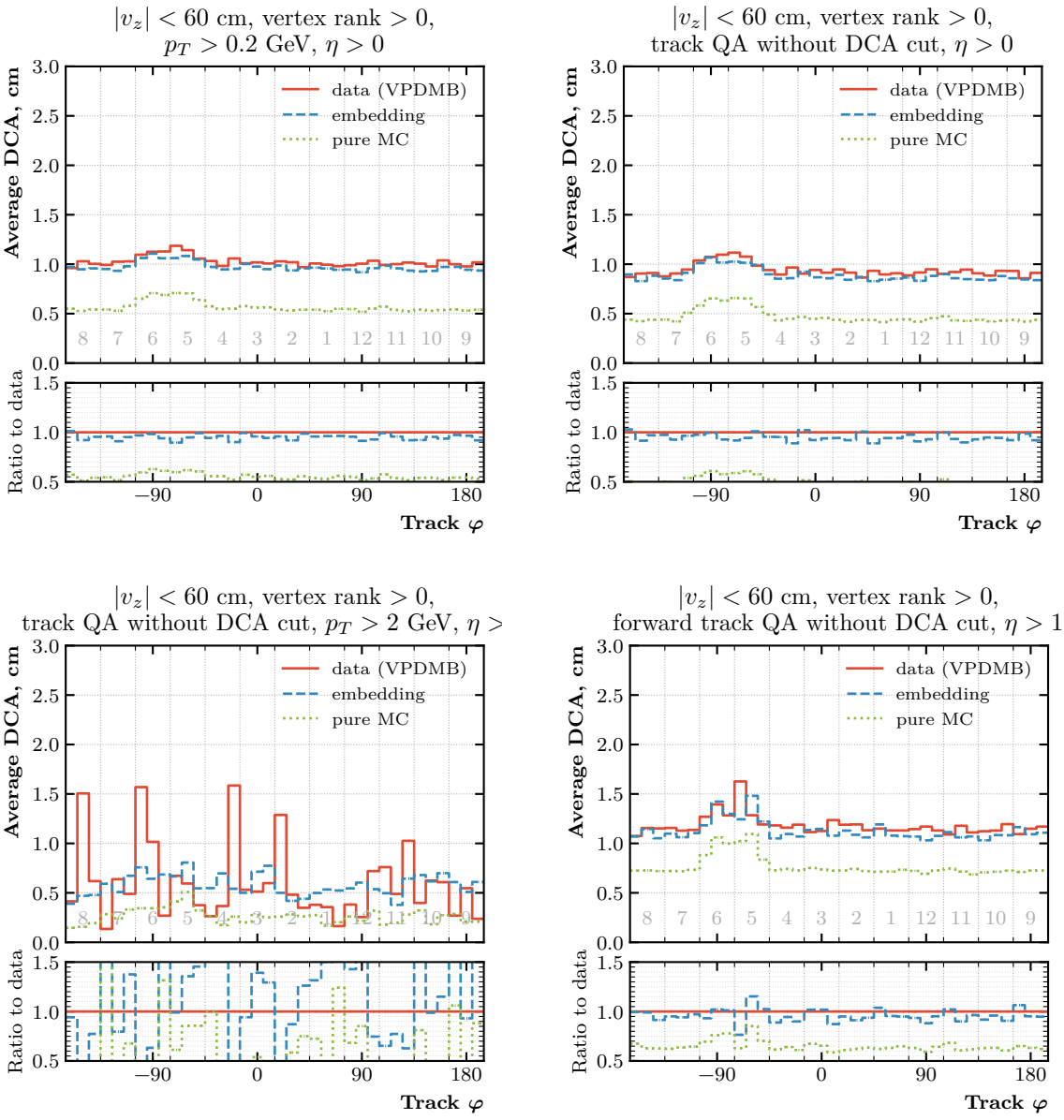

Figure 27: Average track DCA for different track ϕ compared between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

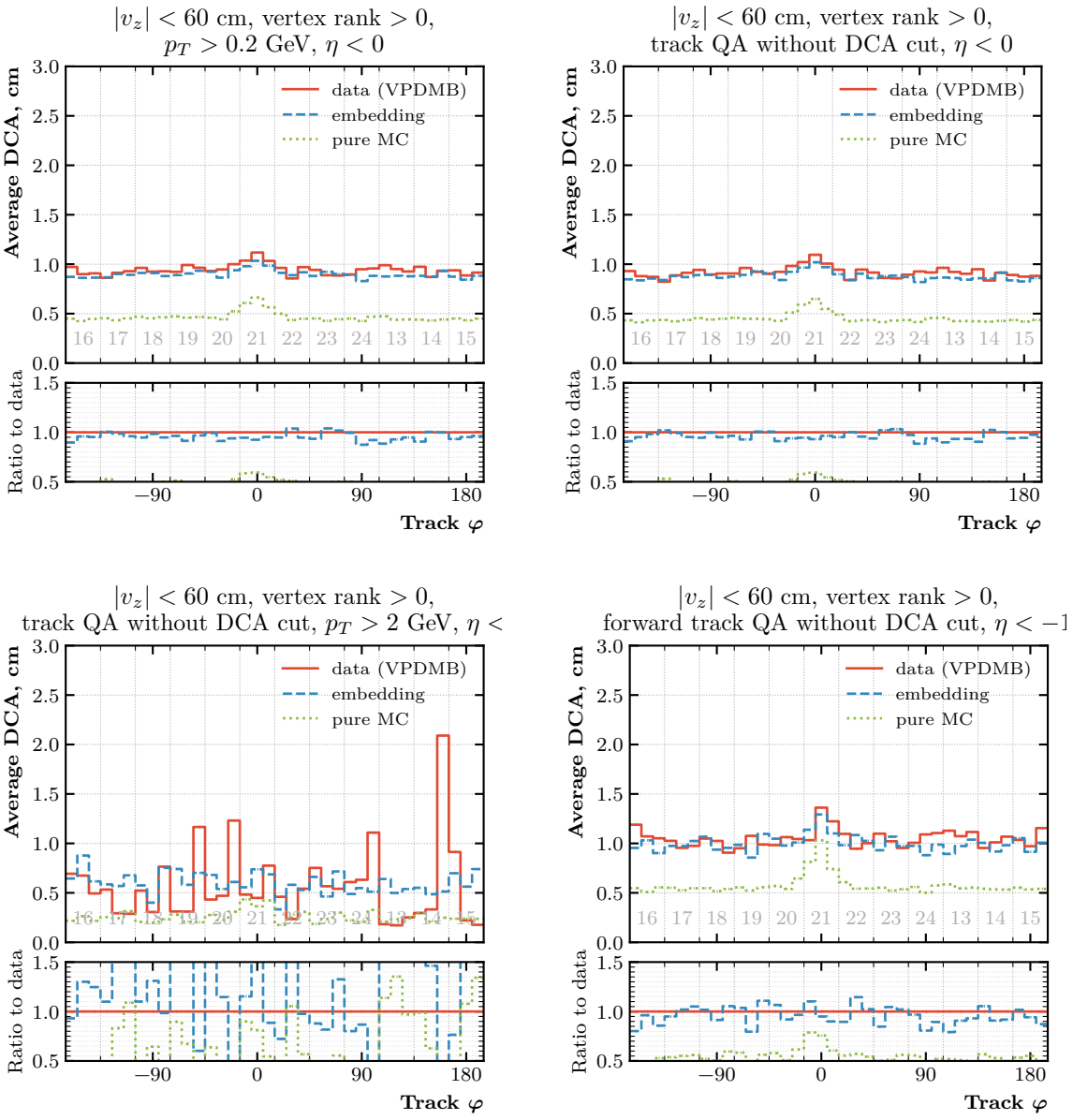

Figure 28: Average track DCA for different track ϕ compared between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

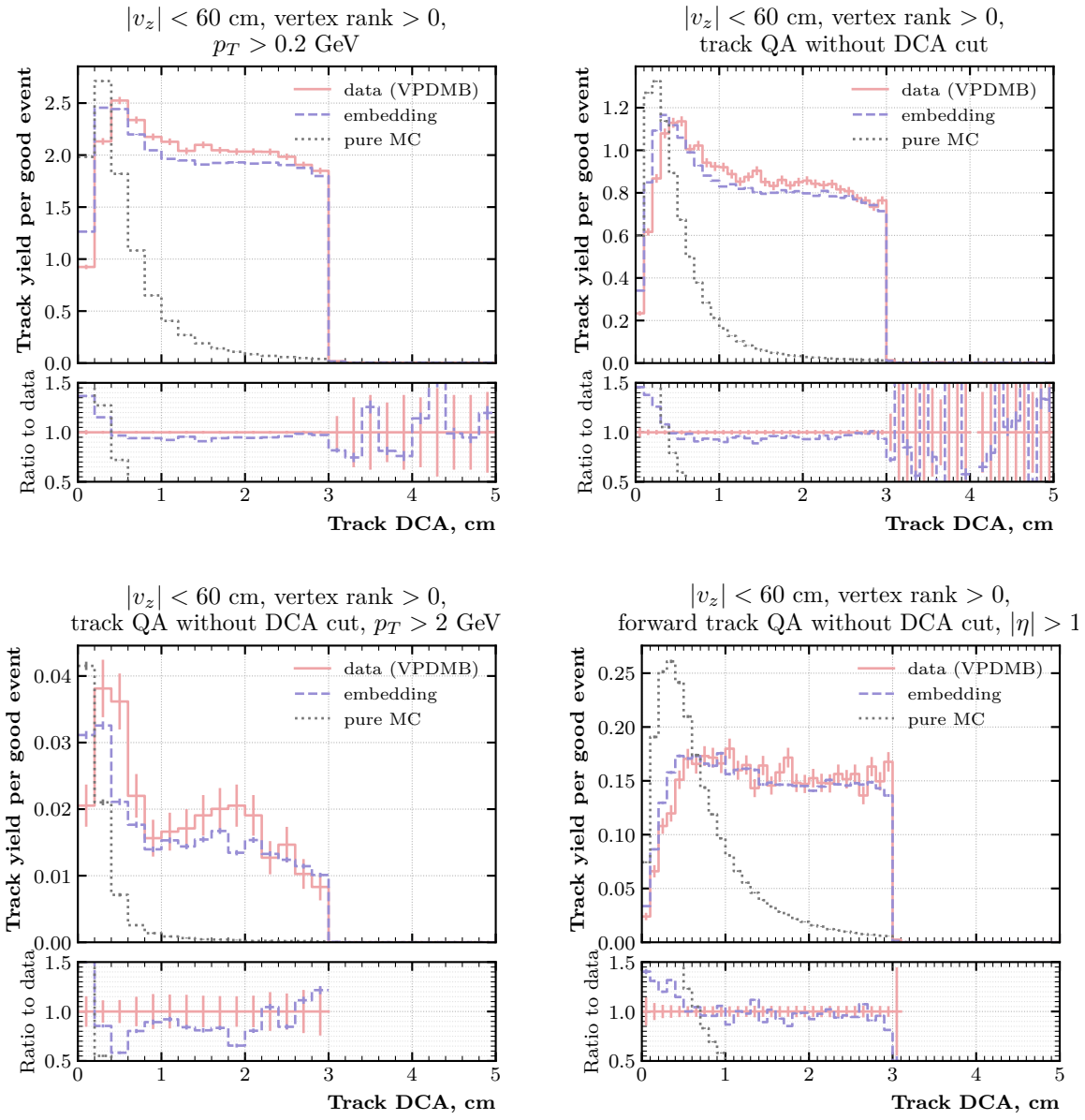

Figure 29: Track DCA distributions comparison between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

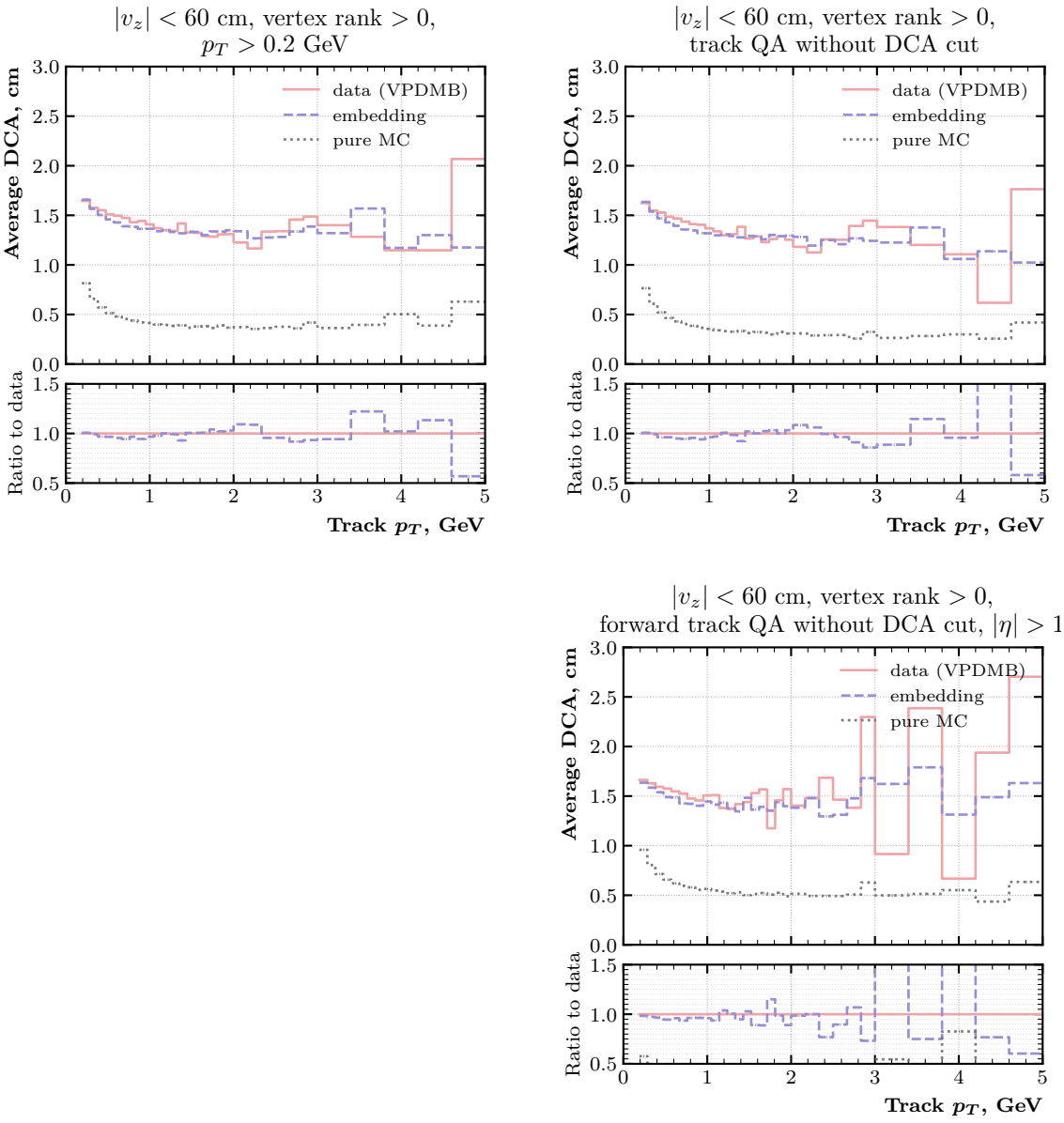

Figure 30: Average track DCA for different track p_T compared between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

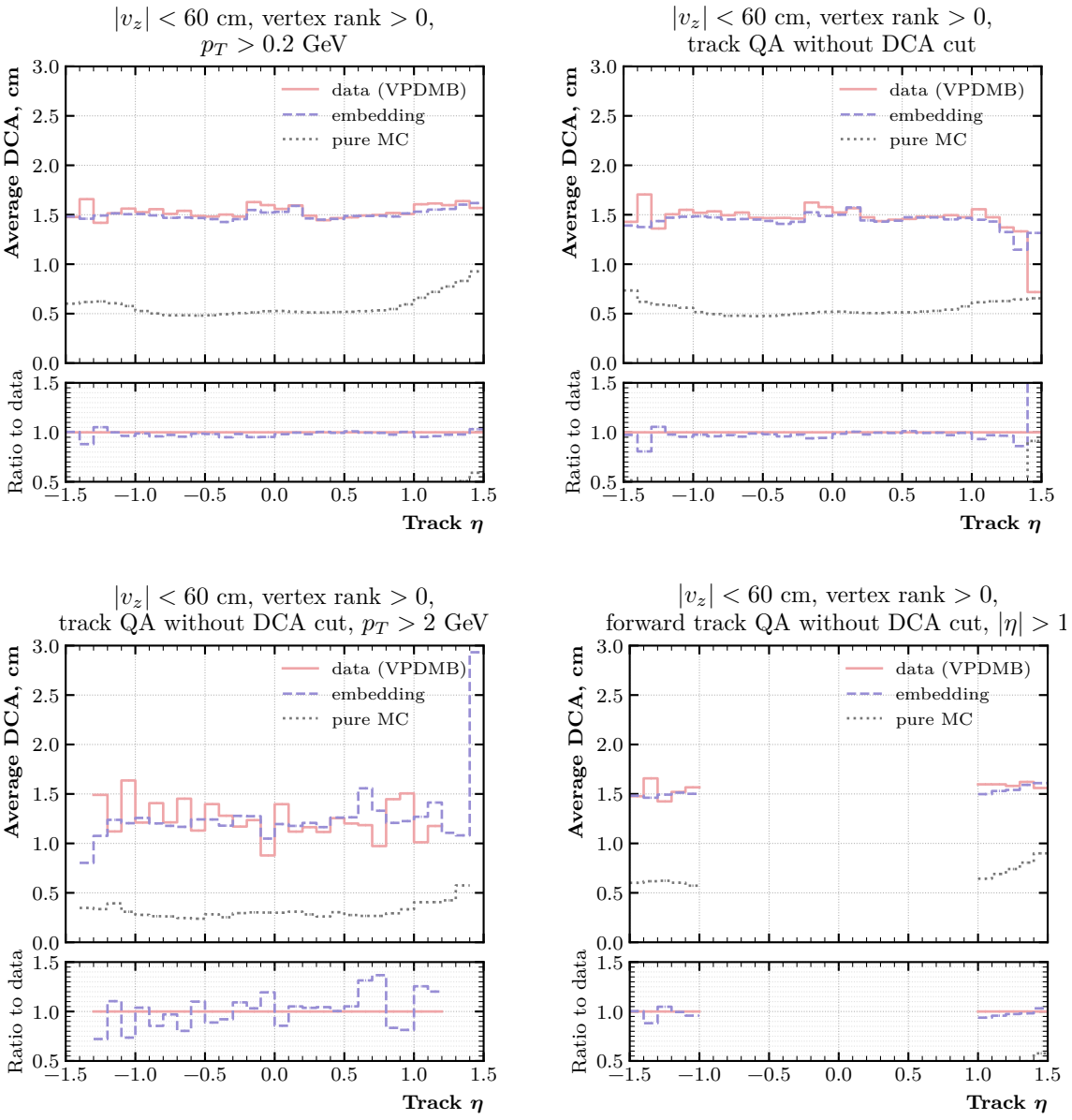

Figure 31: Average track DCA for different track η compared between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

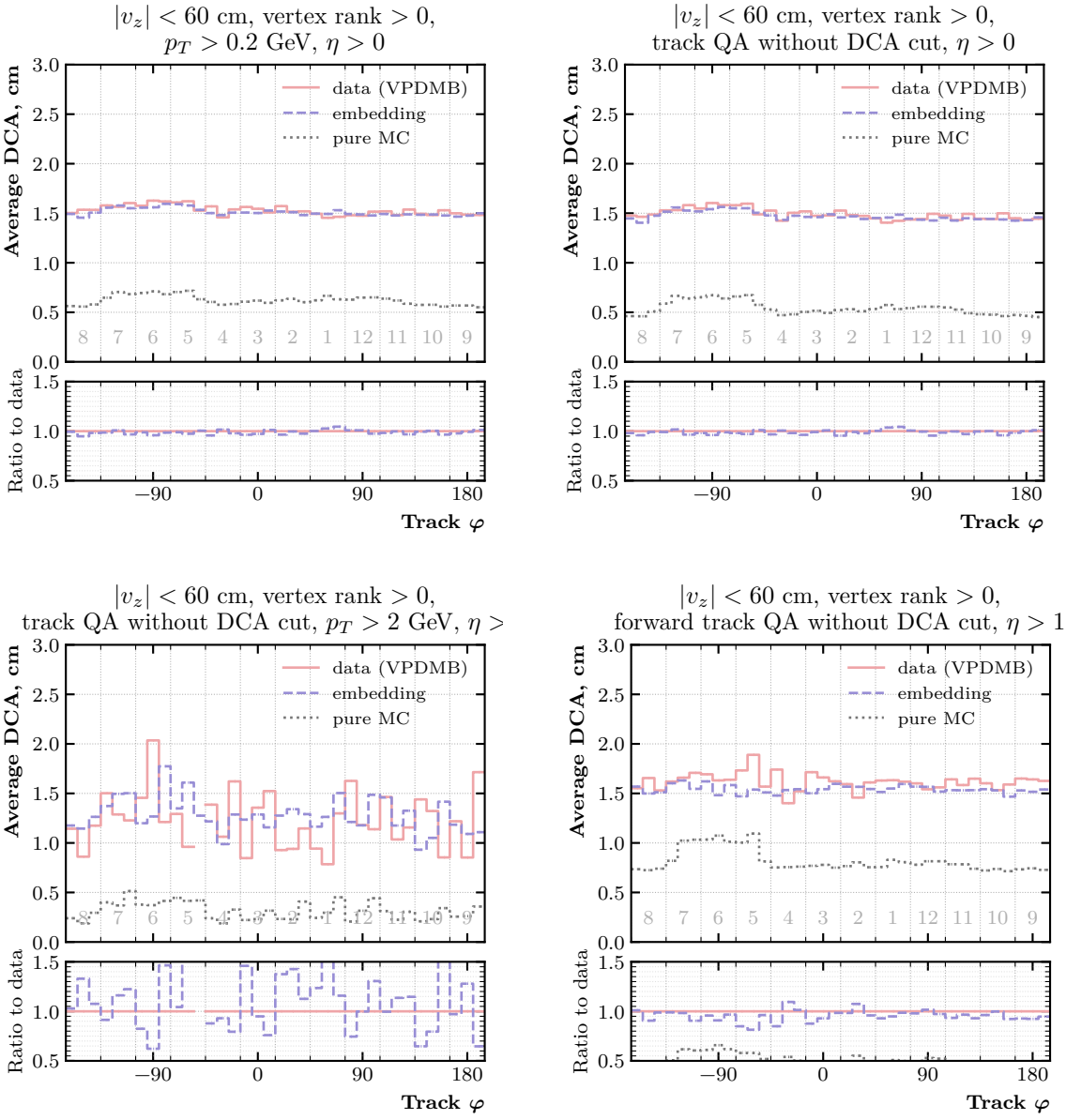

Figure 32: Average track DCA for different track ϕ compared between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

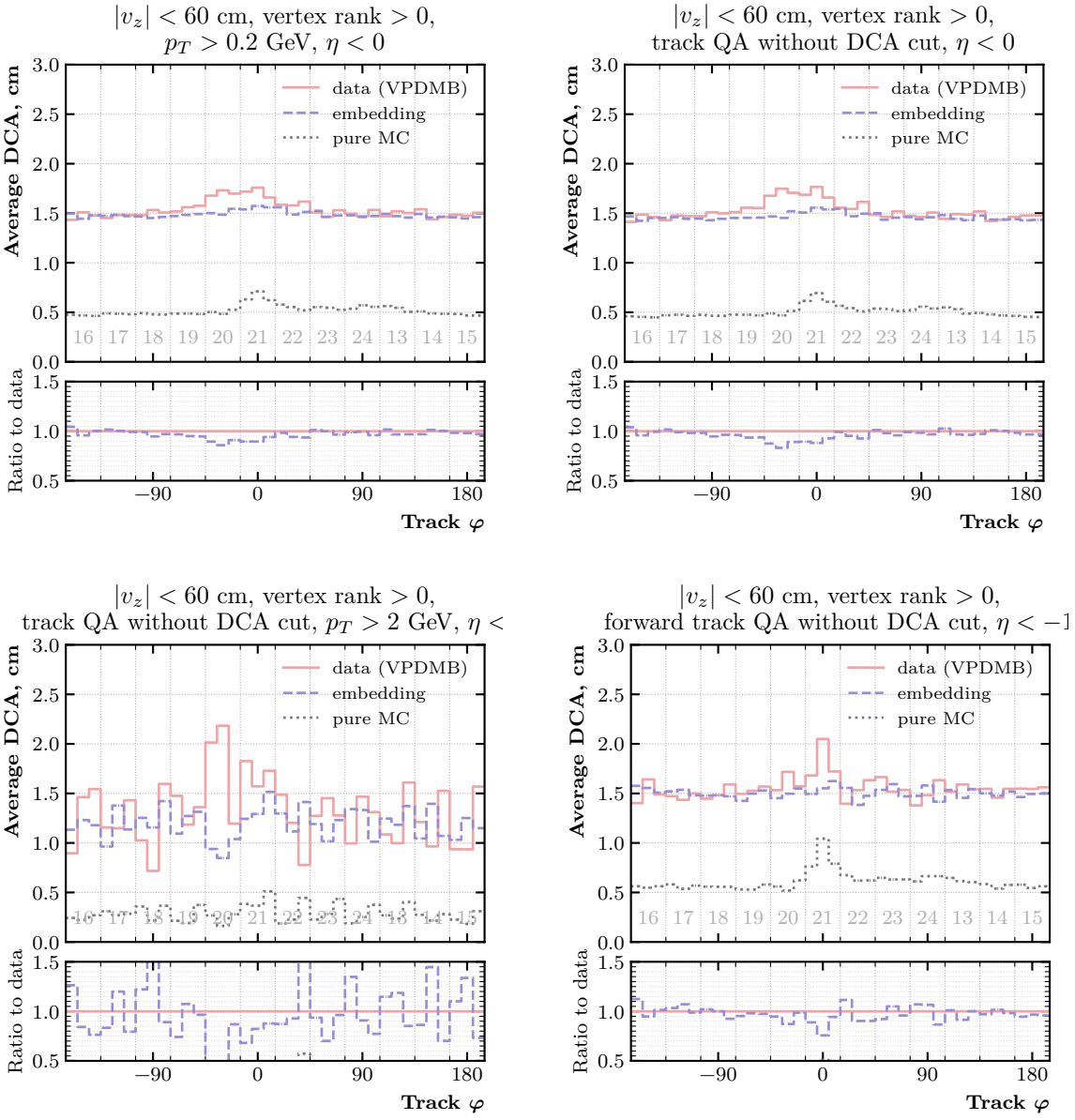

Figure 33: Average track DCA for different track ϕ compared between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

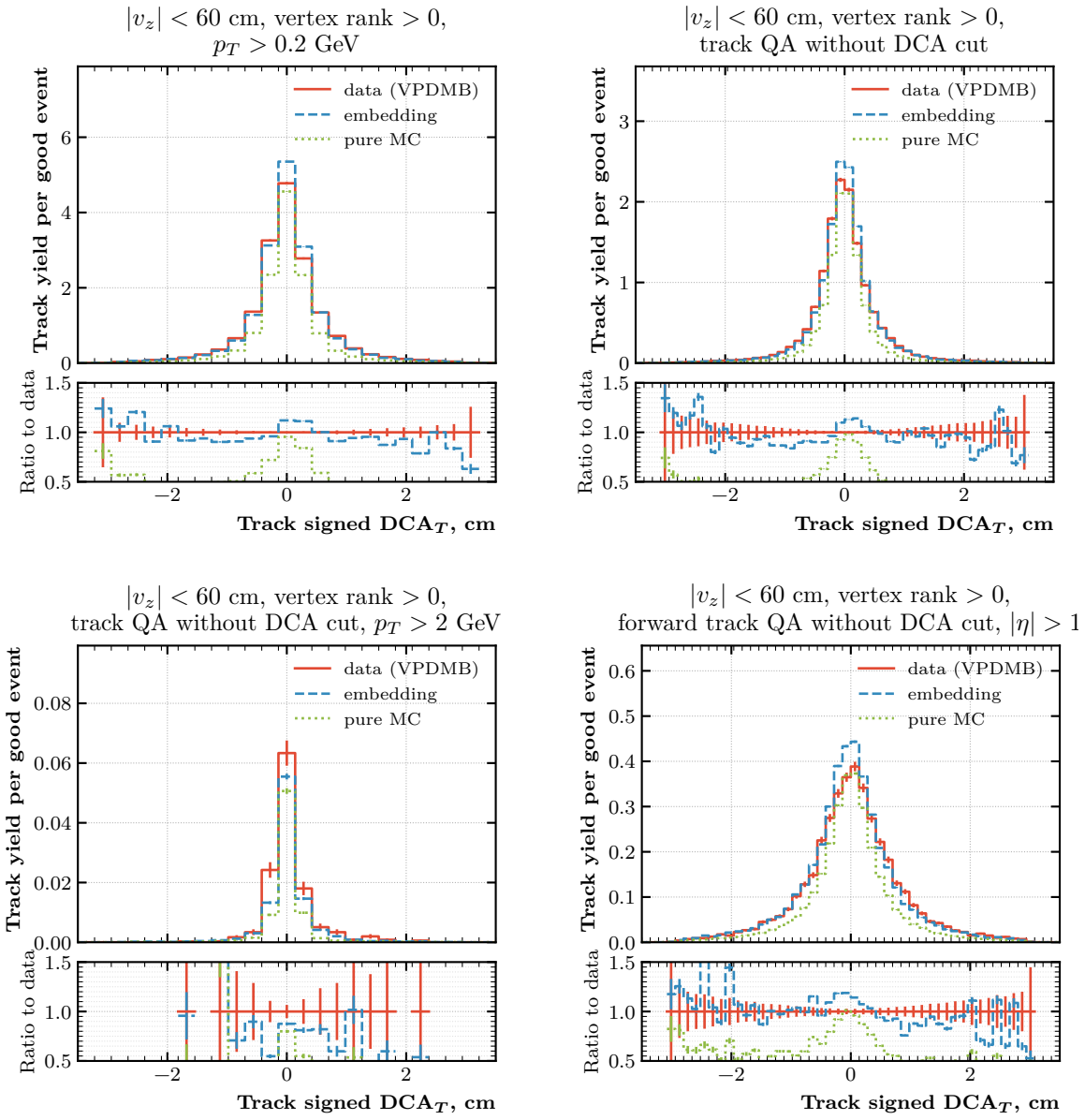

Figure 34: Track signed DCA distributions comparison between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

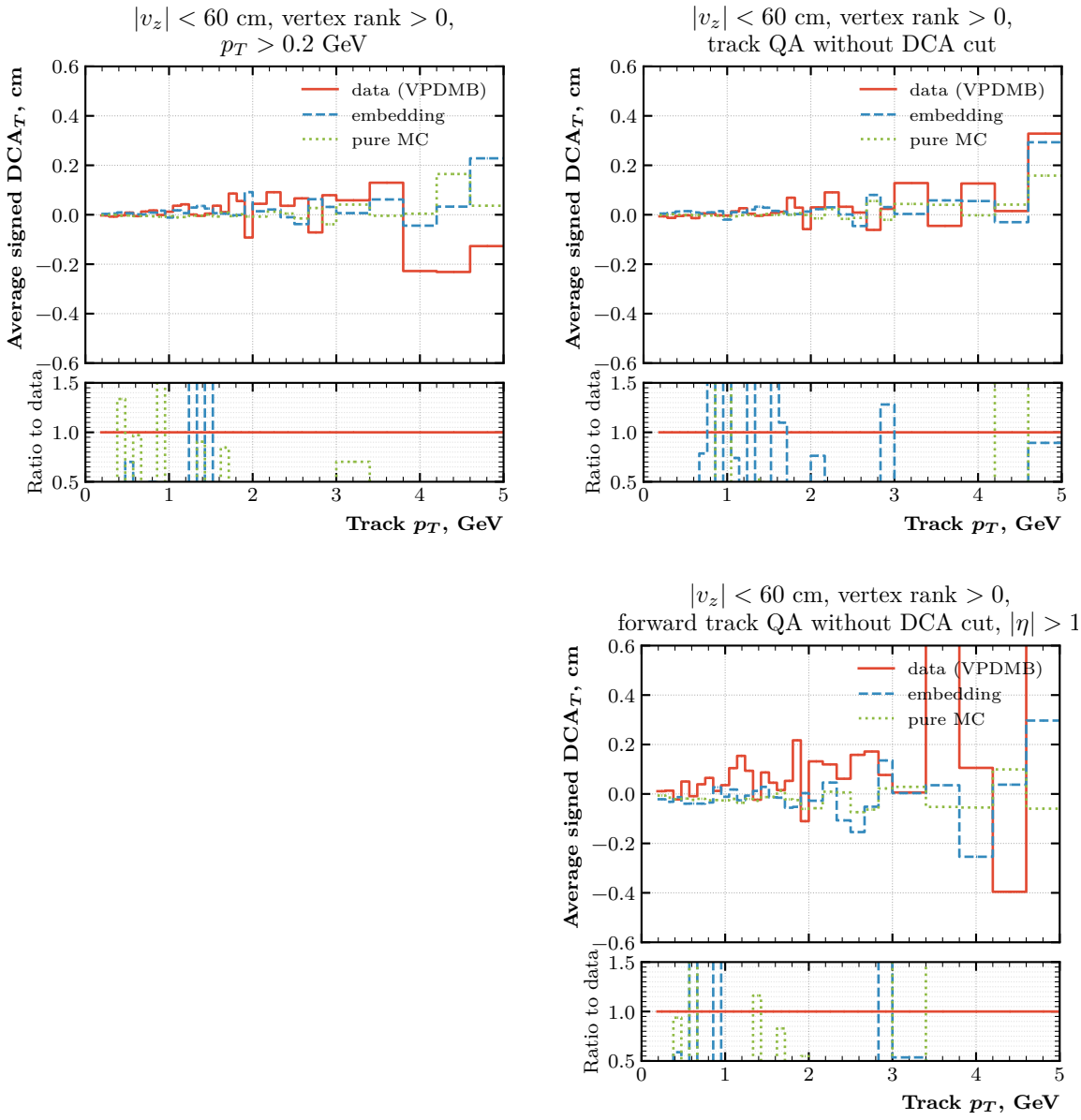

Figure 35: Average track signed DCA for different track p_T compared between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

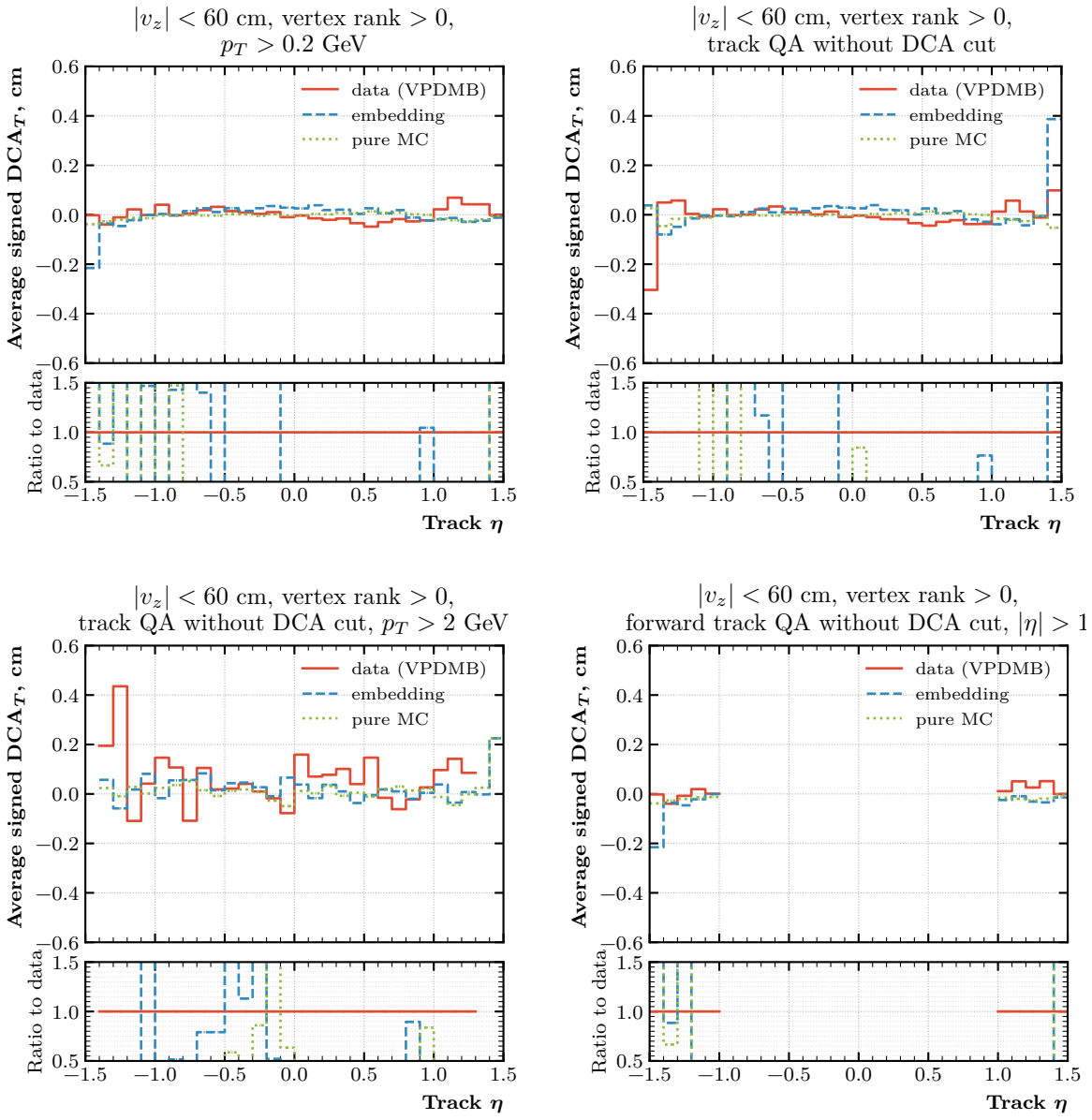

Figure 36: Average track signed DCA for different track η compared between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

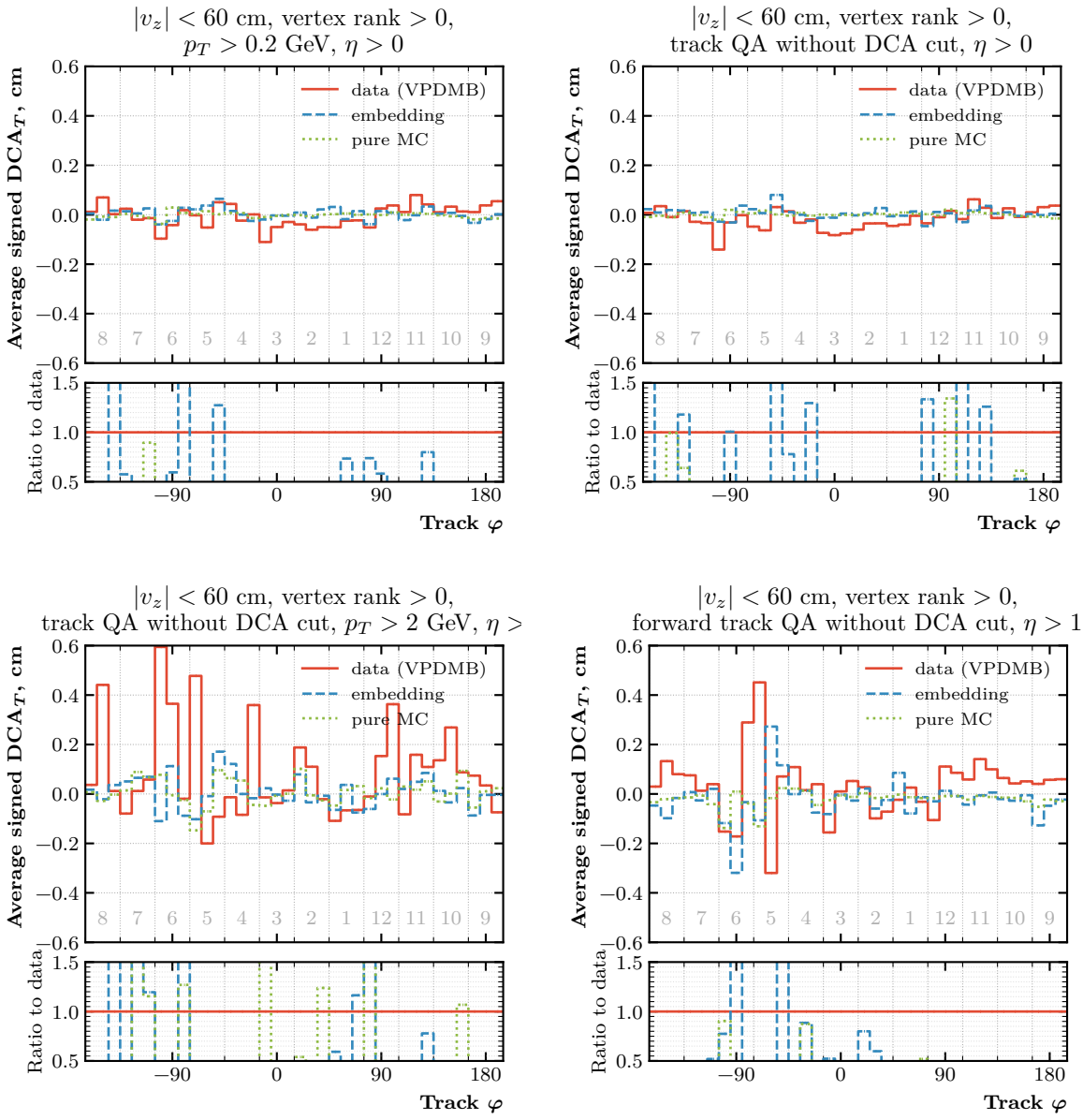

Figure 37: Average track signed DCA for different track ϕ compared between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

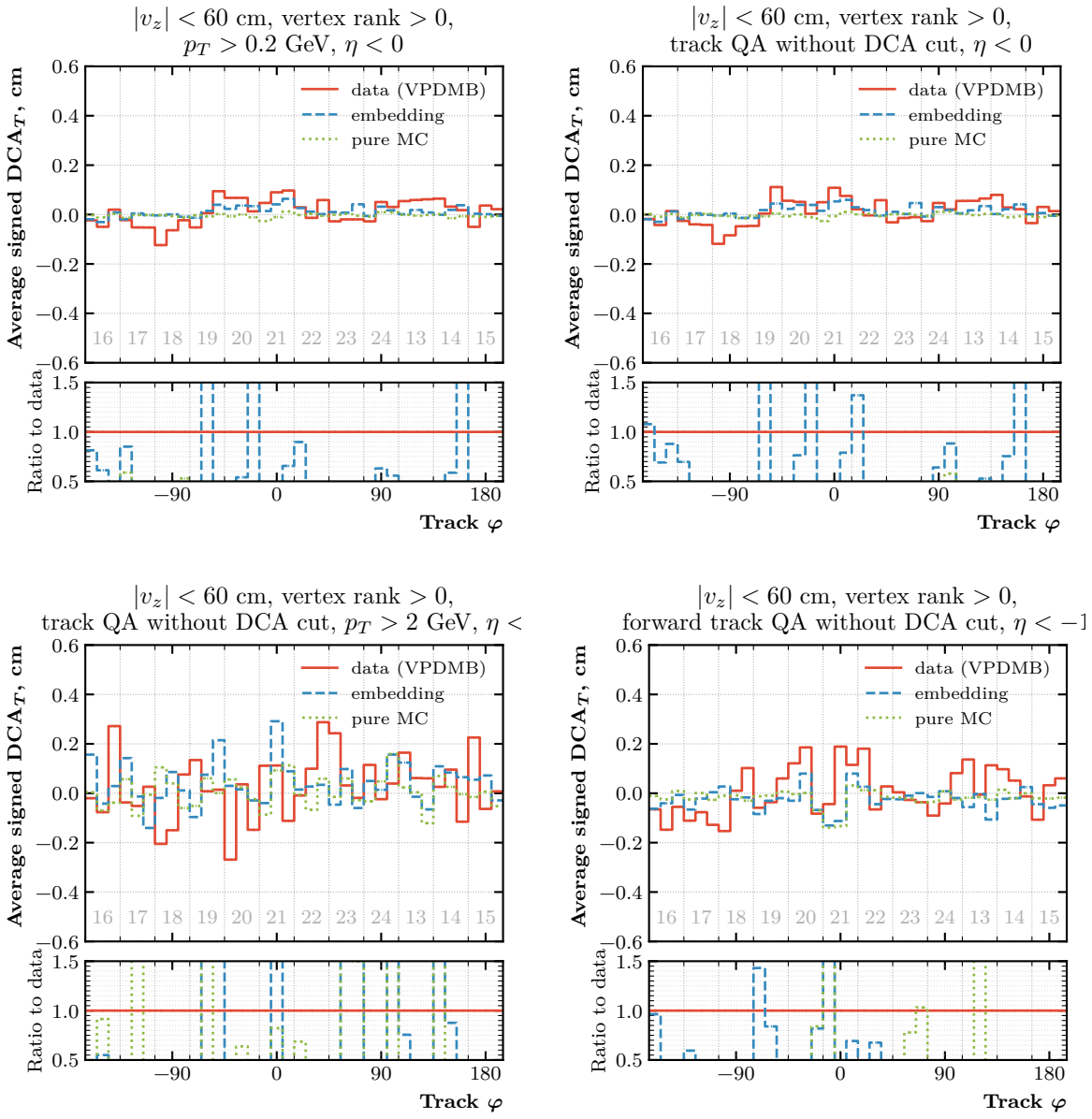

Figure 38: Average track signed DCA for different track ϕ compared between data and simulation for run 13059025 ($\sqrt{s} = 200$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

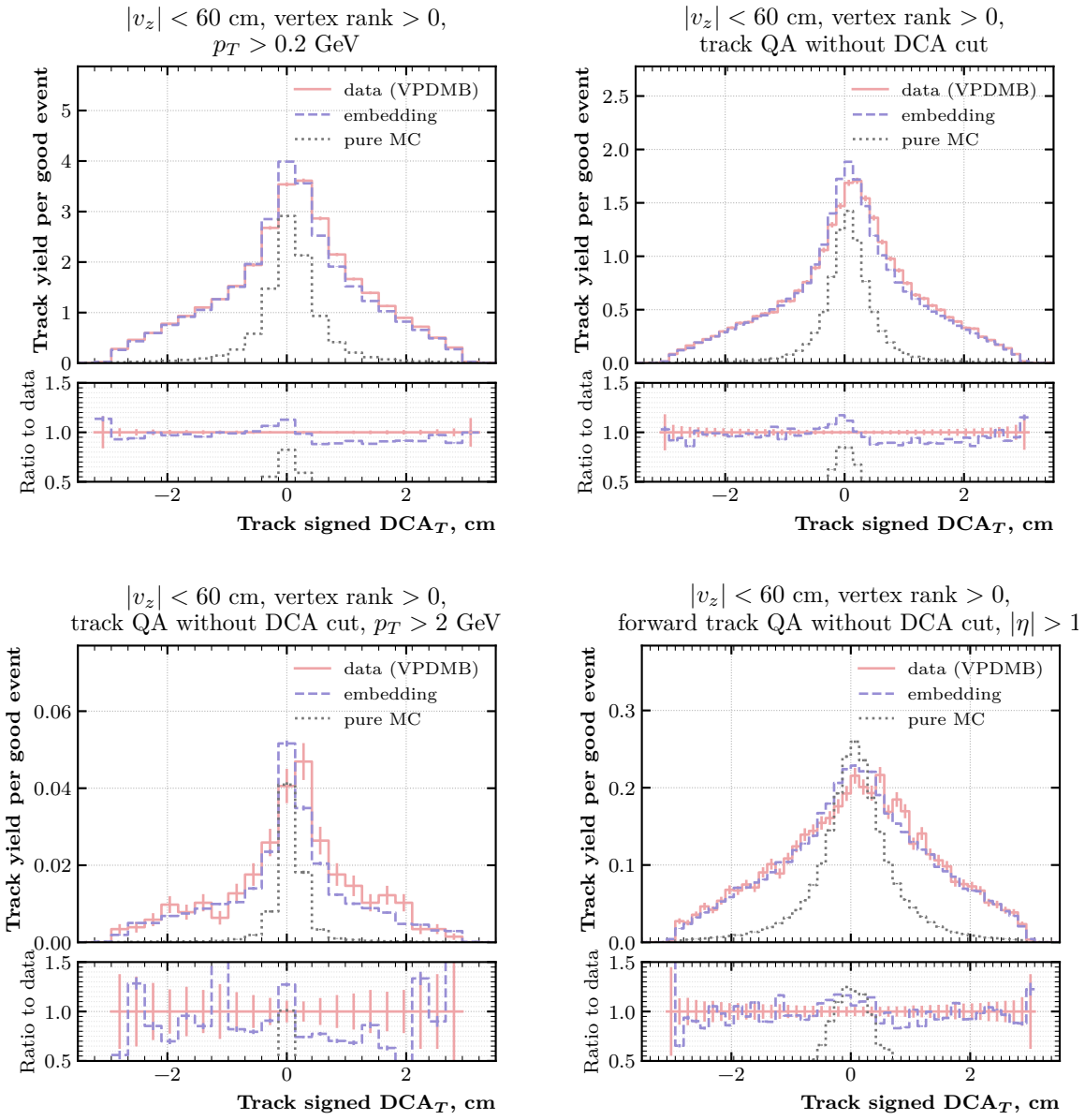

Figure 39: Track signed signed DCA distributions comparison between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

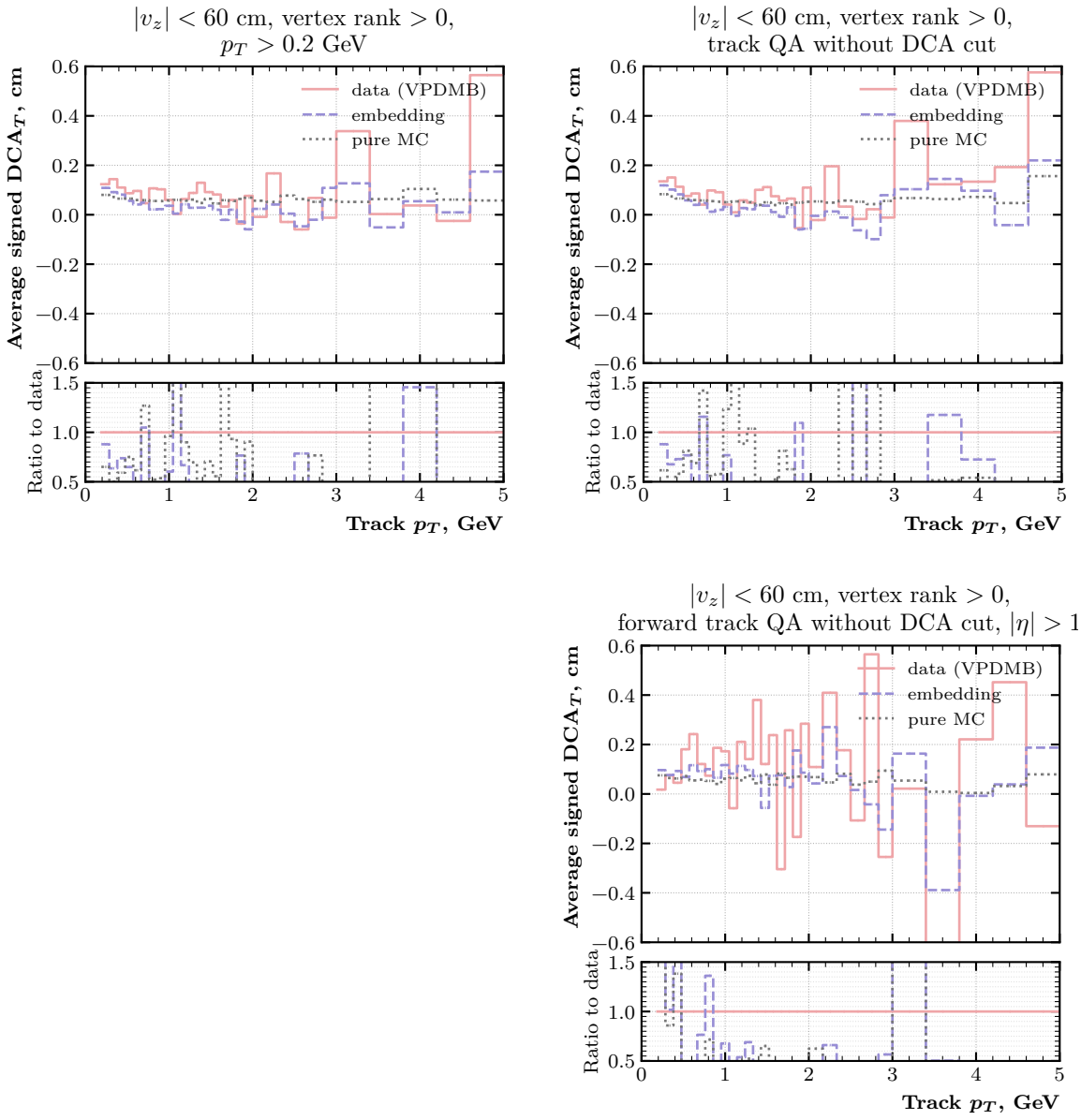

Figure 40: Average track signed DCA for different track p_T compared between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

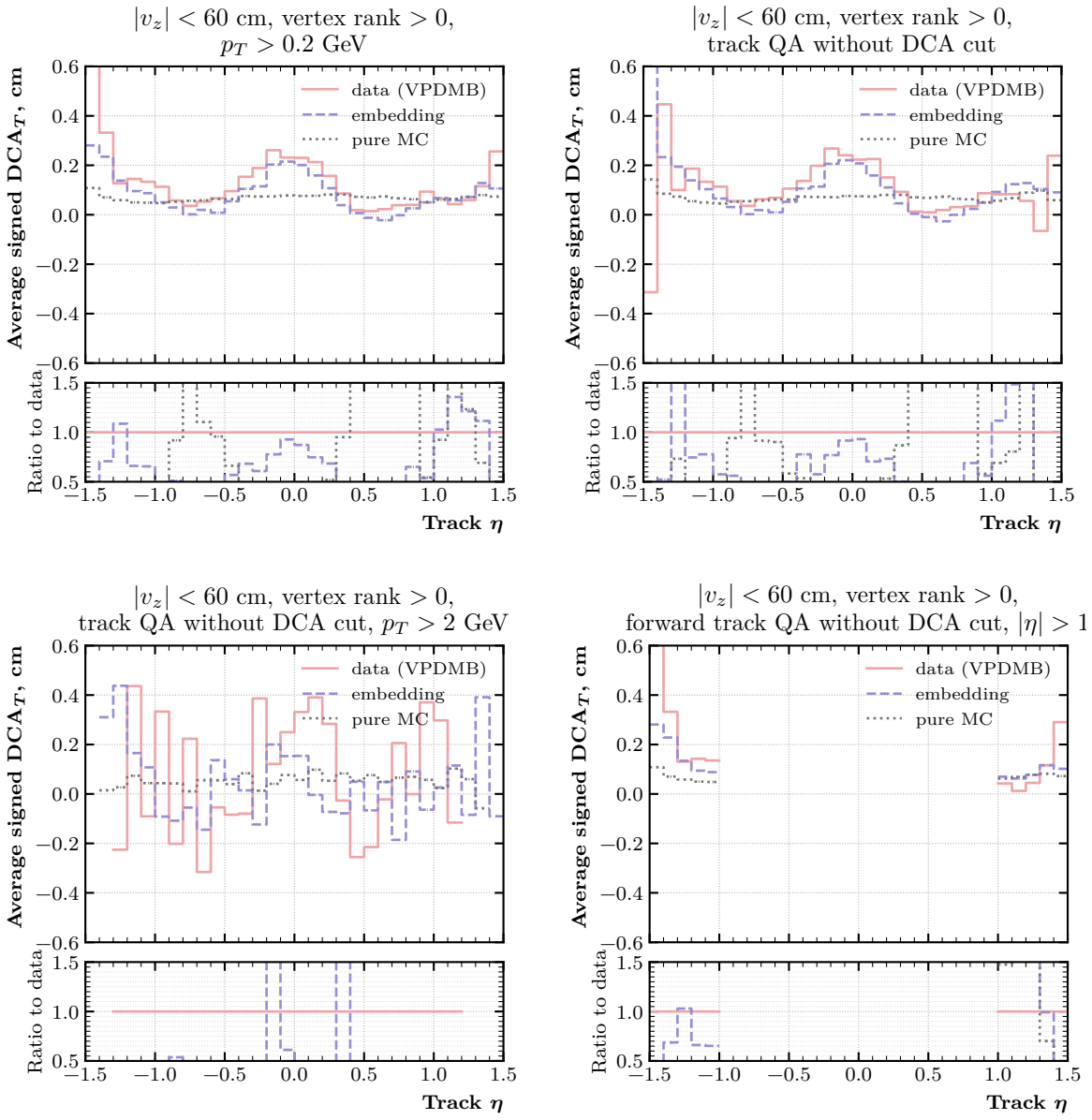

Figure 41: Average track signed DCA for different track η compared between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

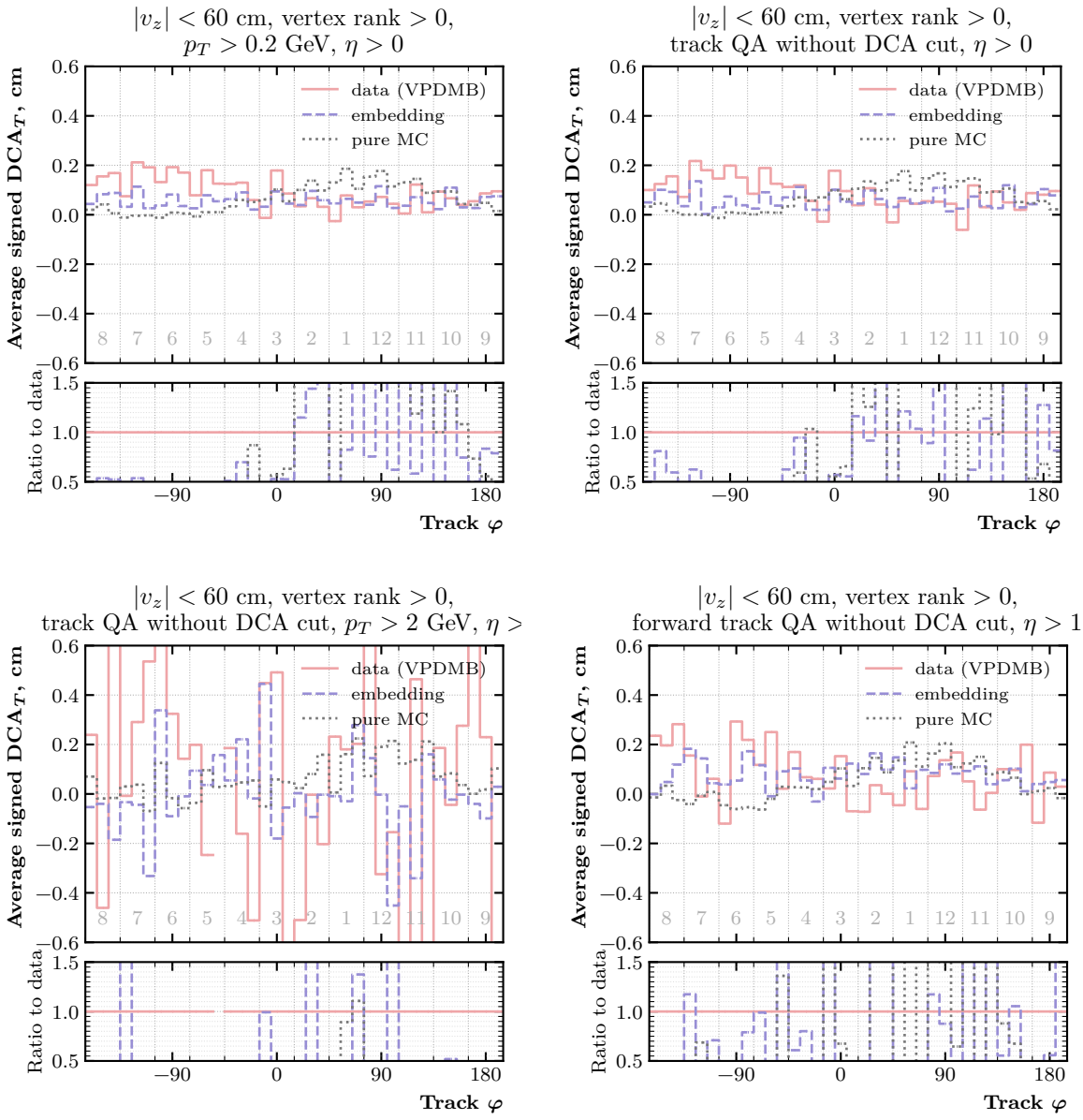

Figure 42: Average track signed DCA for different track ϕ compared between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

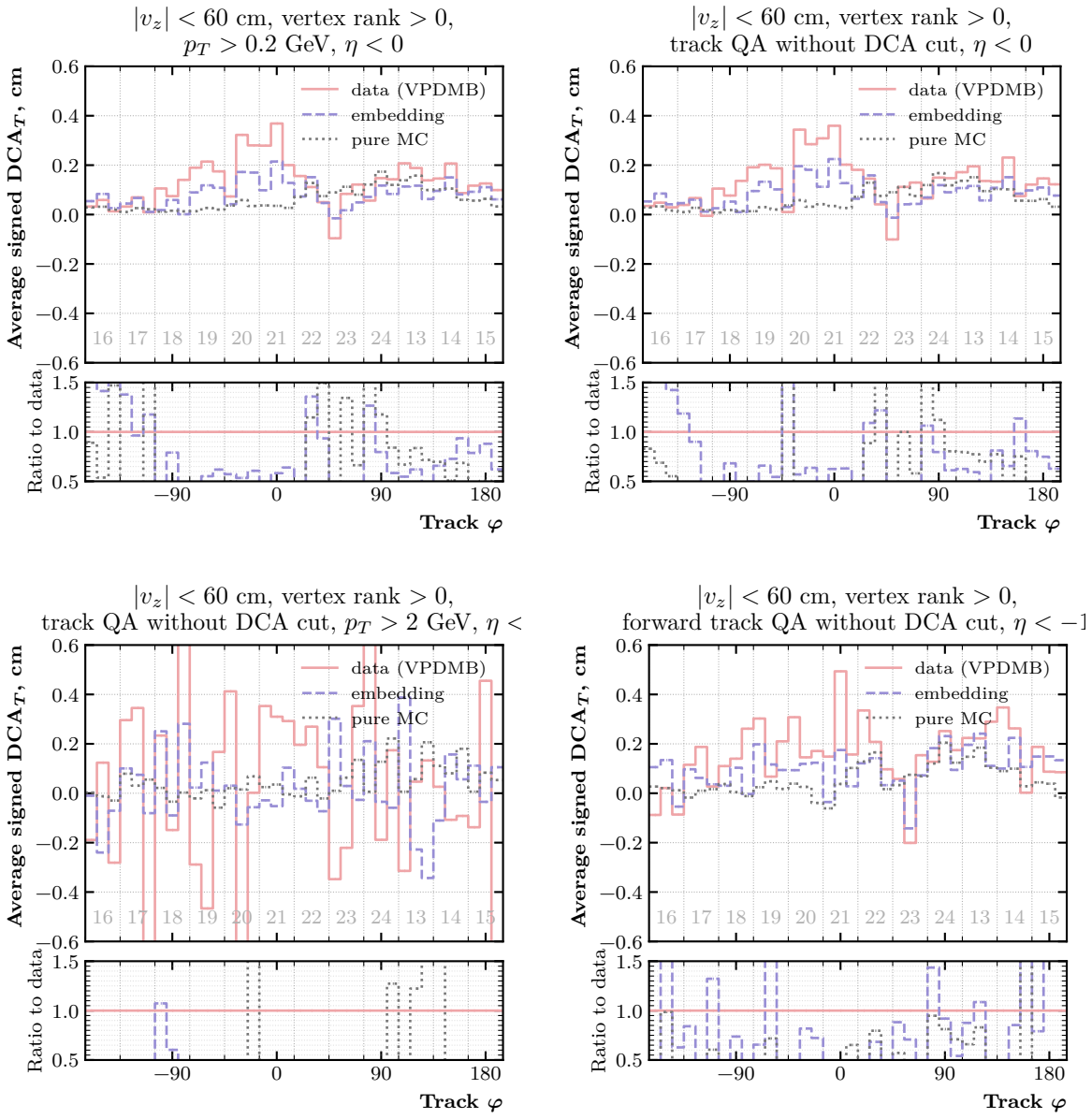

Figure 43: Average track signed DCA for different track ϕ compared between data and simulation for run 13104003 ($\sqrt{s} = 510$ GeV). The yields are normalized to the number of "good events", where a good event is defined as an event with a positively ranking vertex satisfying $|v_z| < 60$ cm. No normalization to the bin width has been applied.

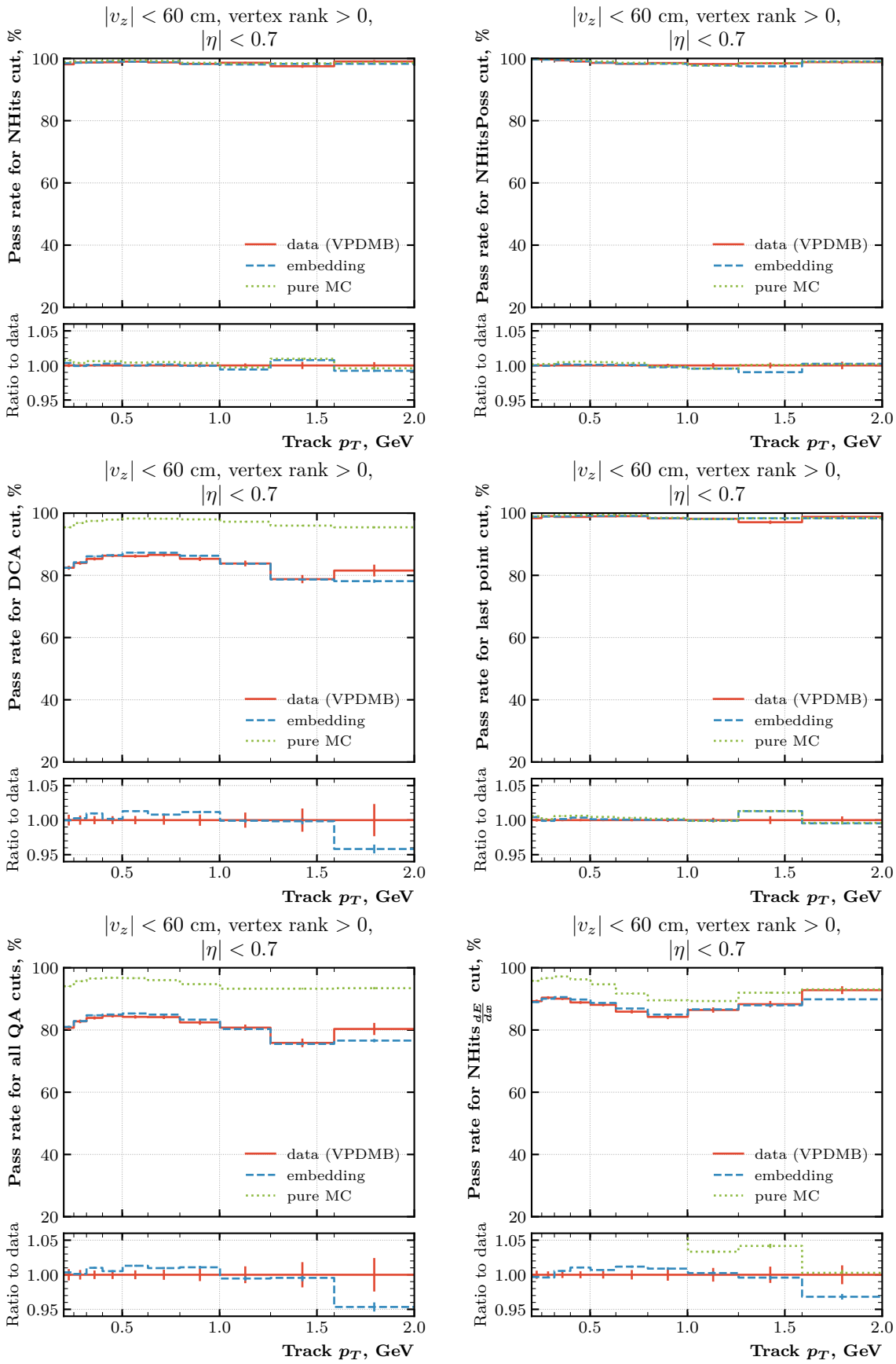

Figure 44: Fraction of tracks that pass a given track QA cut vs track p_T in run 13059025 ($\sqrt{s} = 200$ GeV). The cut on NHits$_{dE/dx}$ is not included in "QA cuts".

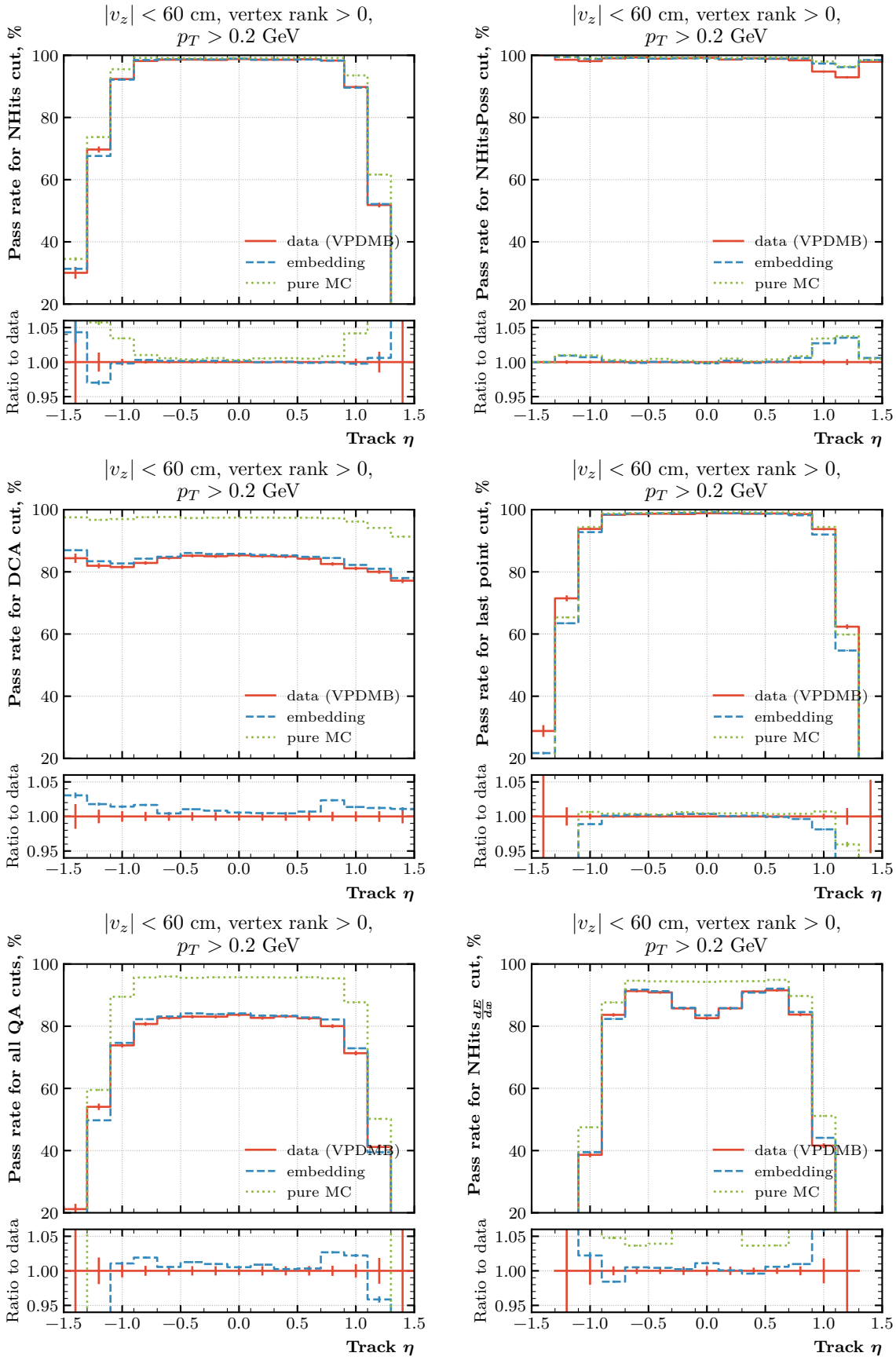

Figure 45: Fraction of tracks that pass a given track QA cut vs track η in run 13059025 ($\sqrt{s} = 200$ GeV). The cut on NHits$_{dE/dx}$ is not included in "QA cuts".

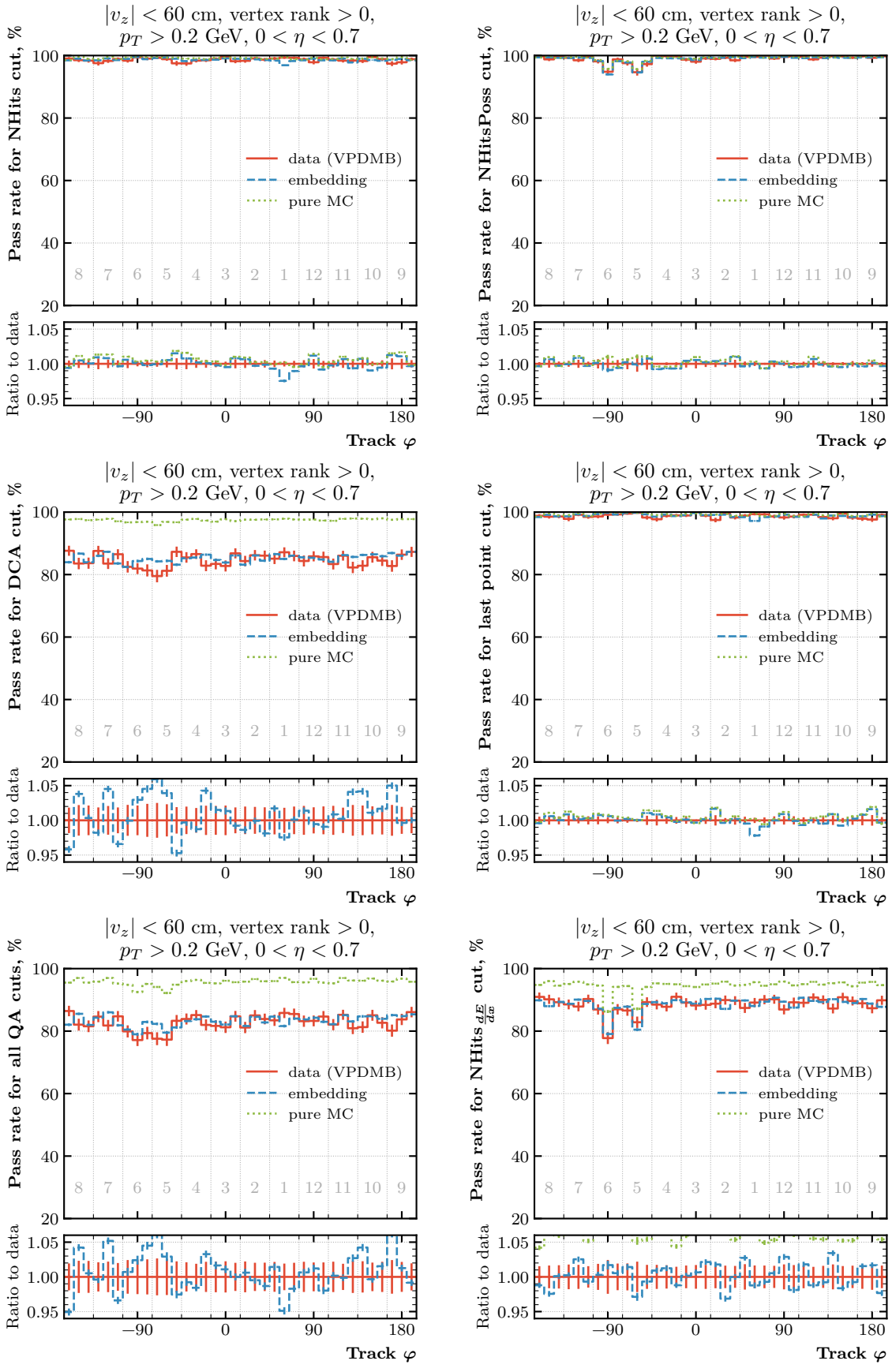

Figure 46: Fraction of tracks that pass a given track QA cut vs track ϕ in run 13059025 ($\sqrt{s} = 200$ GeV). The cut on NHits$_{dE/dx}$ is not included in "QA cuts".

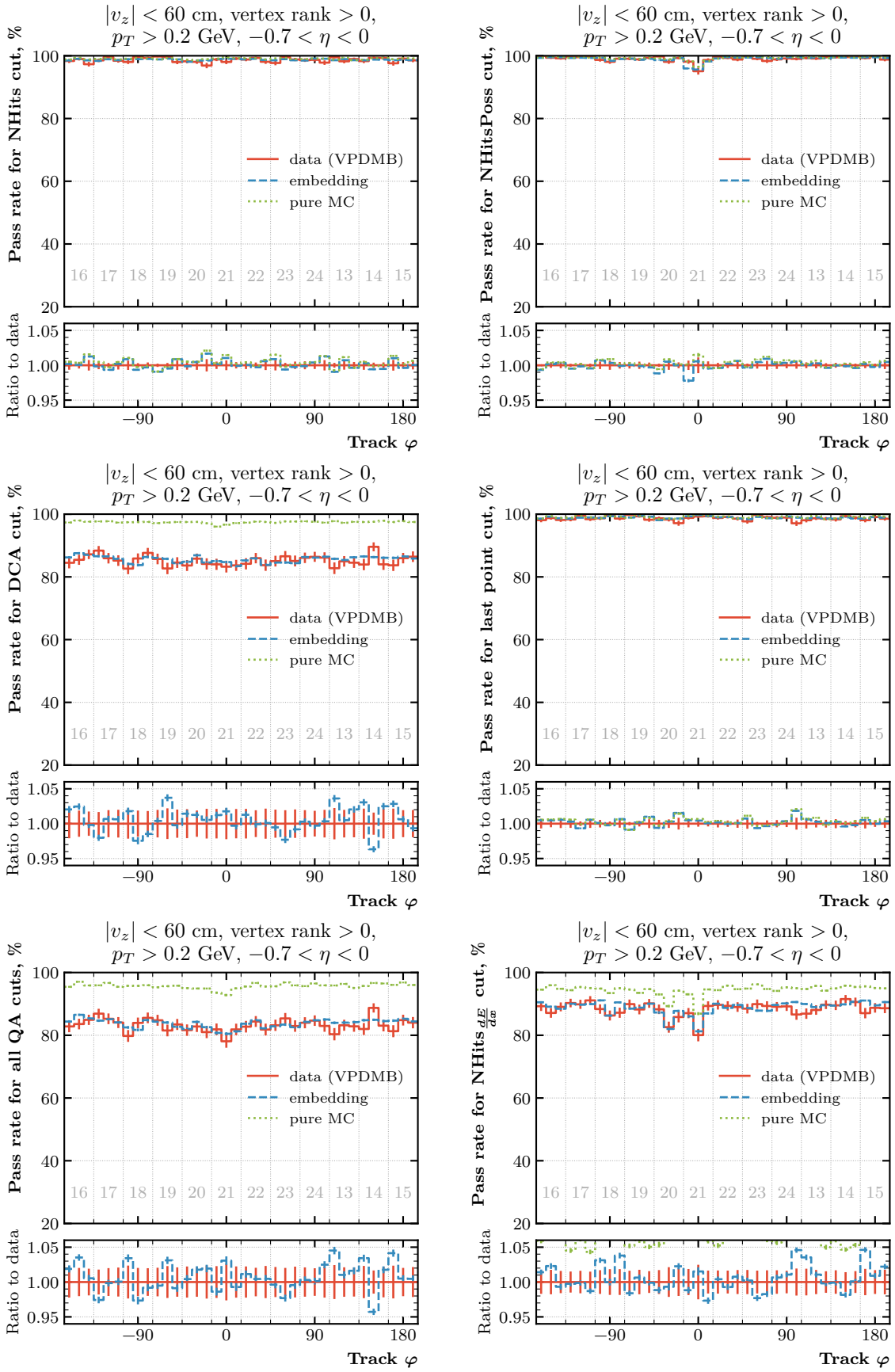

Figure 47: Fraction of tracks that pass a given track QA cut vs track ϕ in run 13059025 ($\sqrt{s} = 200$ GeV). The cut on NHits$_{dE/dx}$ is not included in "QA cuts".

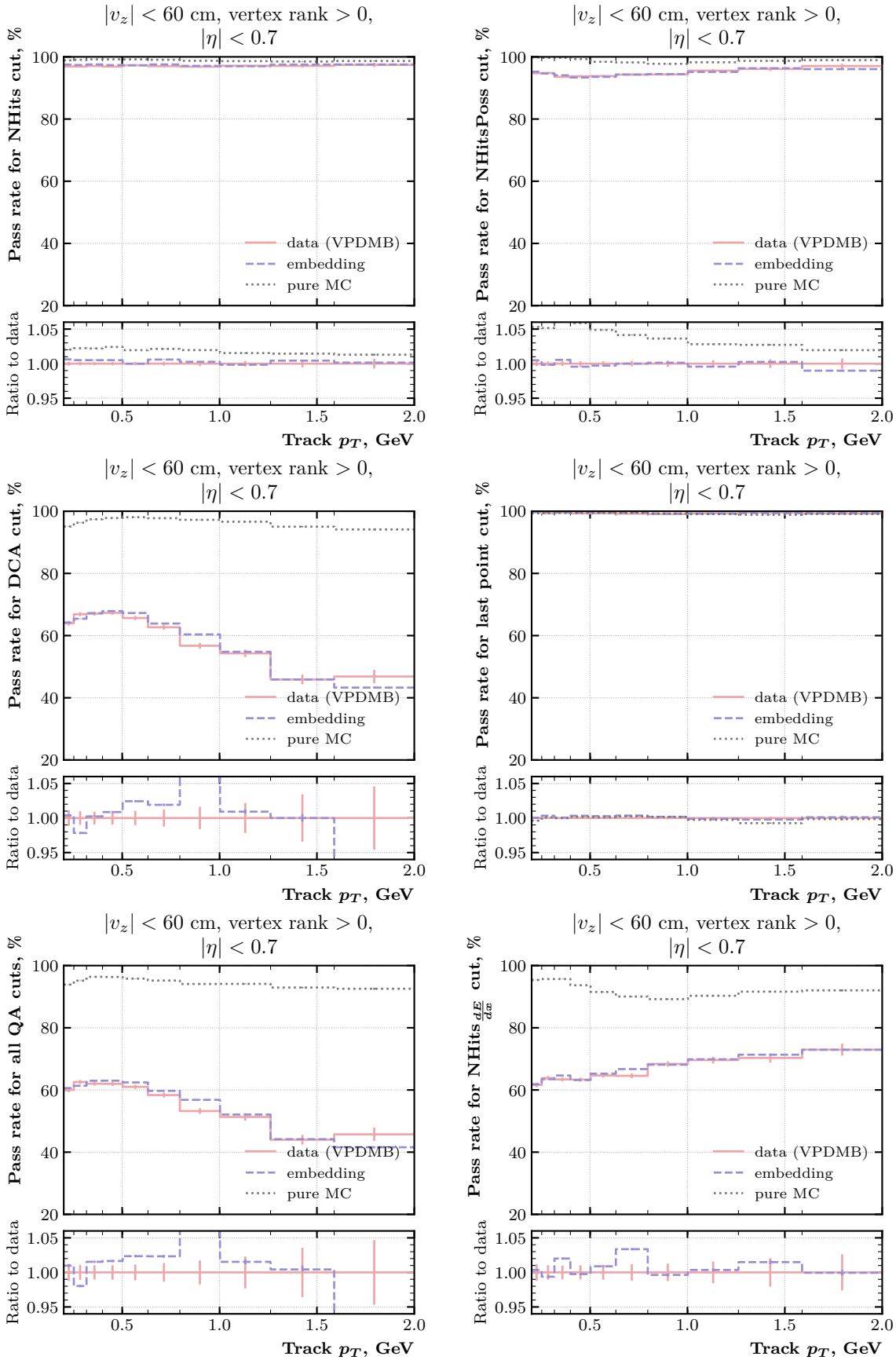

Figure 48: Fraction of tracks that pass a given track QA cut vs track p_T in run 13104003 ($\sqrt{s} = 510$ GeV). The cut on NHits$_{dE/dx}$ is not included in "QA cuts".

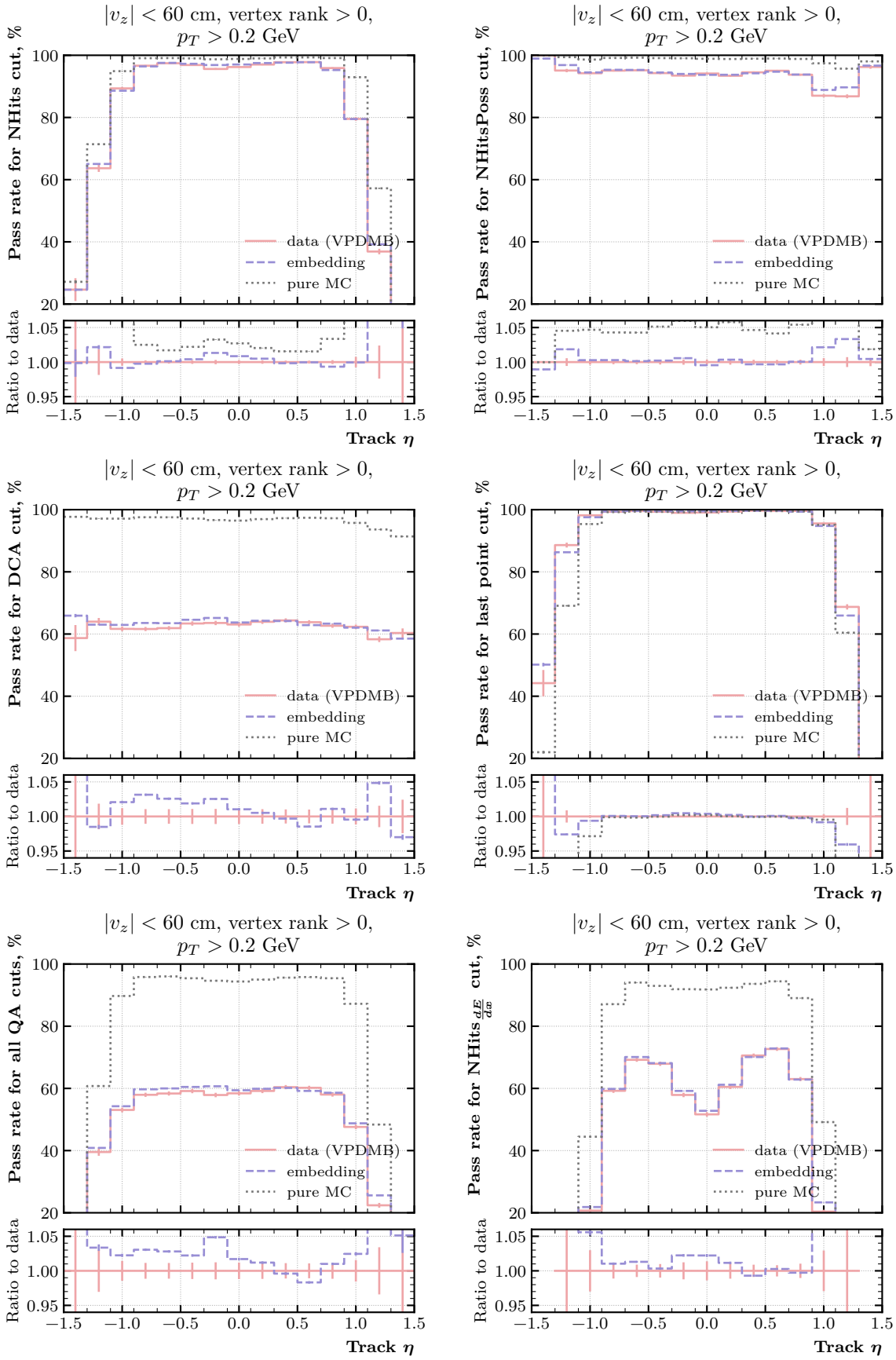

Figure 49: Fraction of tracks that pass a given track QA cut vs track η in run 13104003 ($\sqrt{s} = 510$ GeV). The cut on NHits$_{dE/dx}$ is not included in "QA cuts".

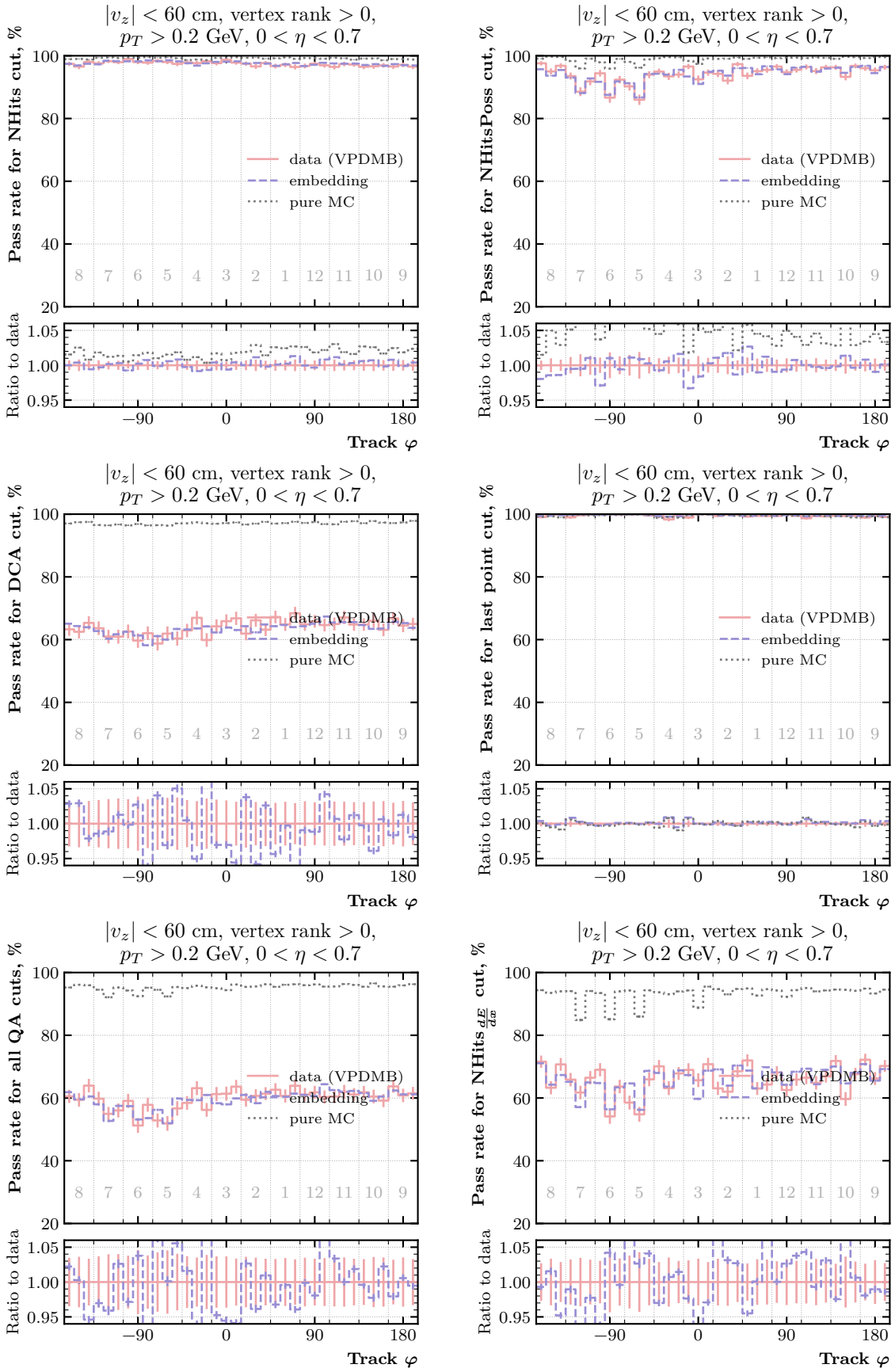

Figure 50: Fraction of tracks that pass a given track QA cut vs track ϕ in run 13104003 ($\sqrt{s} = 510$ GeV). The cut on NHits$_{dE/dx}$ is not included in "QA cuts".

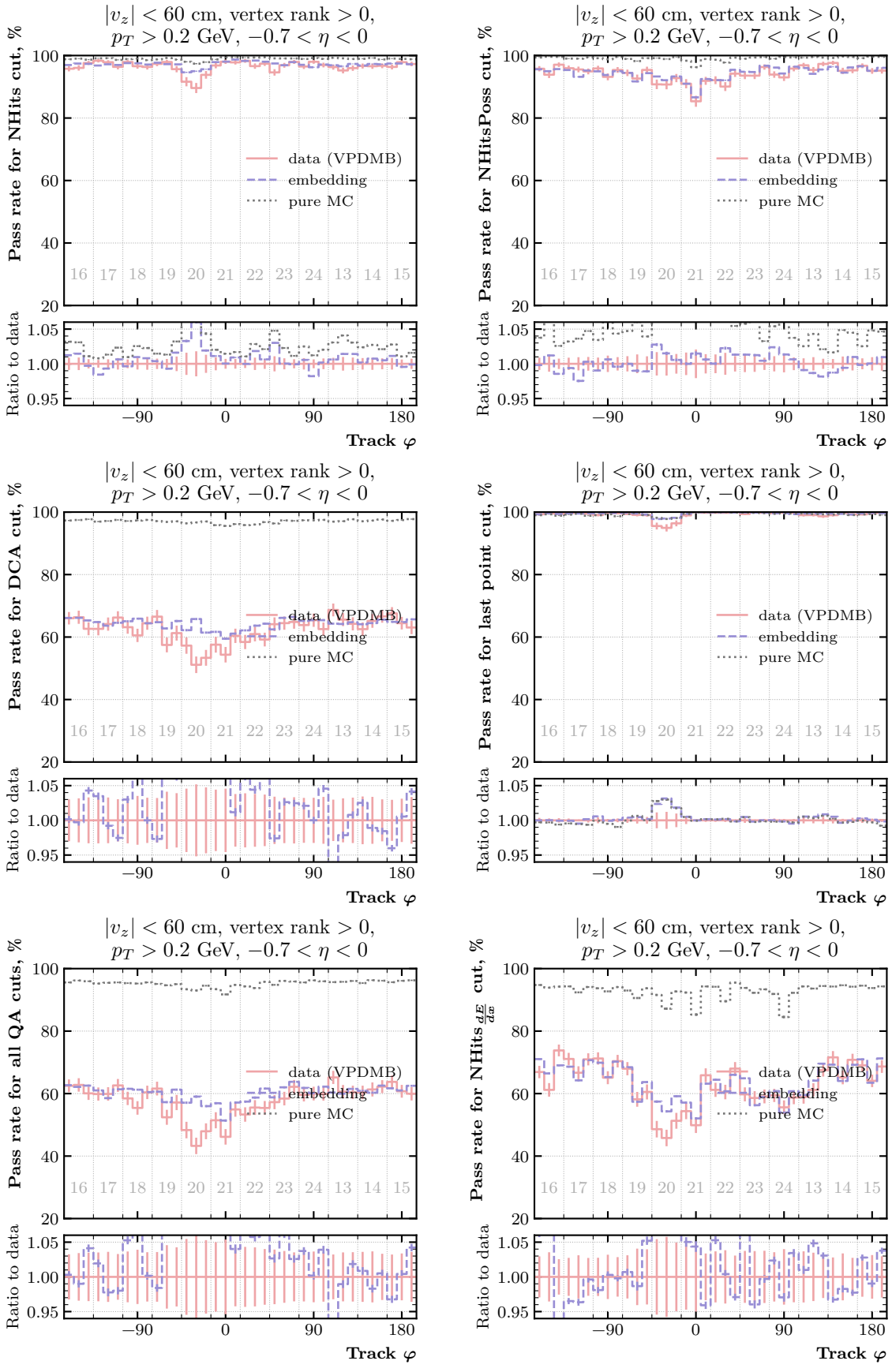

Figure 51: Fraction of tracks that pass a given track QA cut vs track ϕ in run 13104003 ($\sqrt{s} = 510$ GeV). The cut on NHits$_{dE/dx}$ is not included in "QA cuts".

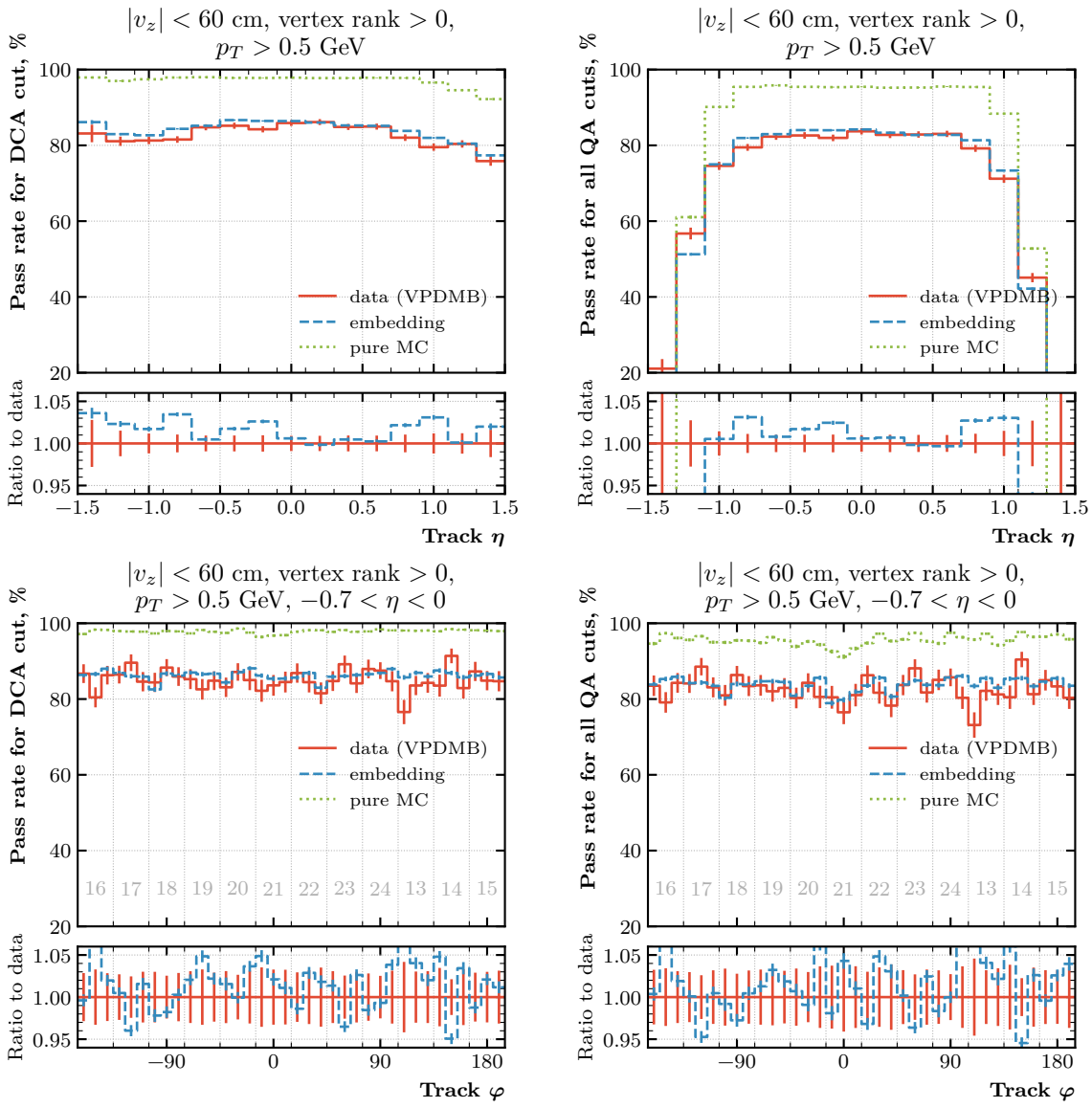

Figure 52: Fraction of tracks that pass a given track QA cut vs track η and track φ in run 13059025 ($\sqrt{s} = 200$ GeV).

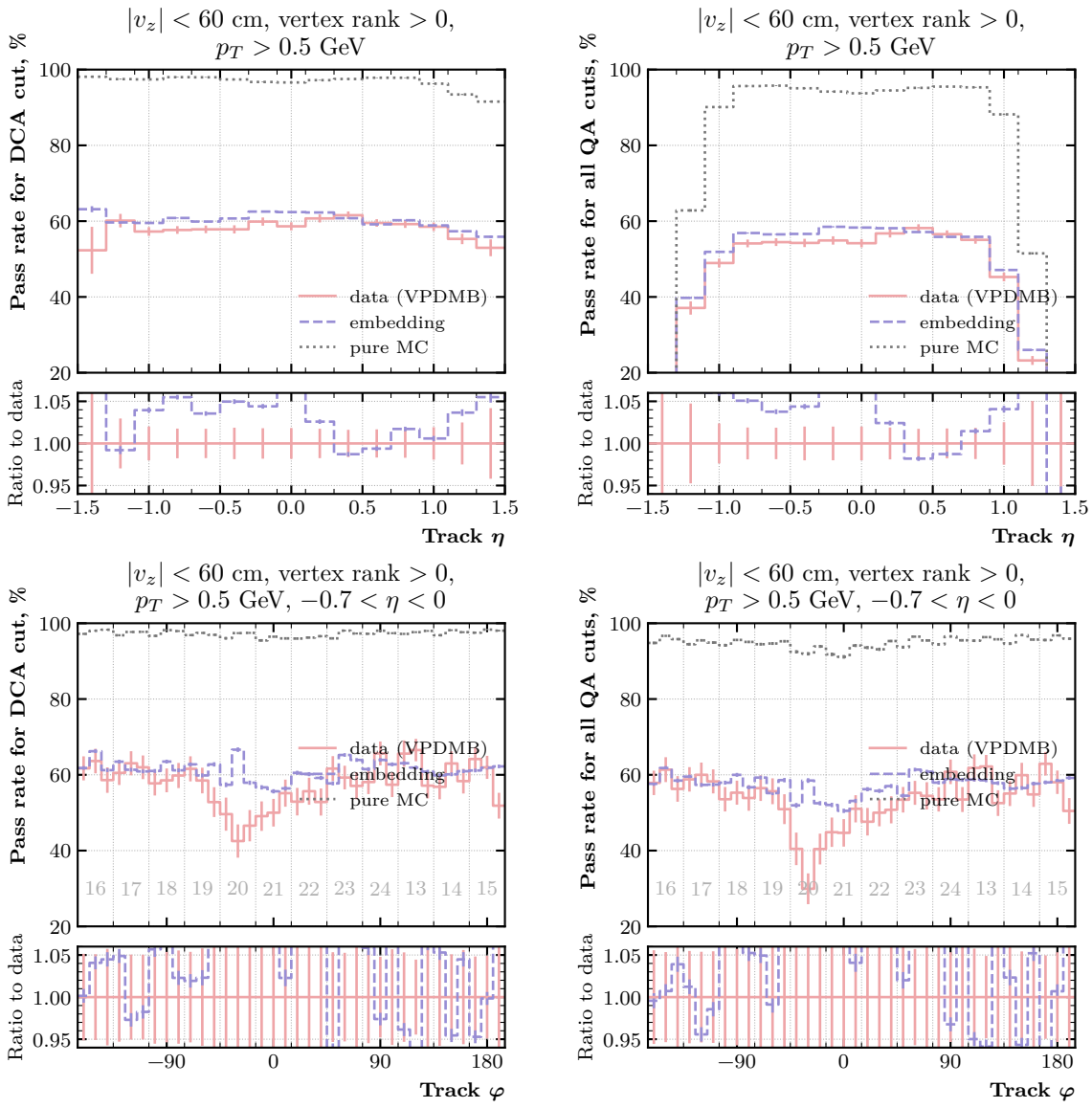

Figure 53: Fraction of tracks that pass a given track QA cut vs track η and track φ in run 13104003 ($\sqrt{s} = 510$ GeV).

7 Conclusion

This article presents a redo of a technically challenging study with single track embedding. The new look with the old approach did not reveal any discrepancies to be visible for the 2012 proton-proton data and simulation. This study also surveyed an extended range of luminosities by looking at the $\sqrt{s} = 200$ GeV and $\sqrt{s} = 510$ GeV subsamples.

The proposed way of studying the tracking efficiency uncertainty using the cut pass rates suggests that the systematic uncertainty can be reduced by at least factor of two compared to the old number of 4%. The specific number for the uncertainty can be better estimated by the analyzer with their full available statistics. This should be a relatively simple thing to do given that it does not involve producing any additional embedding samples in addition to the ones already used in the specific analysis. We believe this task to be outside the scope of the present study.

Recent changes to the STAR simulation framework may break the observed agreement and require further re-tuning. At the time of this writing, this is still an open question.

References

[1] M. Anderson *et al.*, Nucl. Instrum. Meth. A **499**, 659 (2003) doi:10.1016/S0168-9002(02)01964-2 [nucl-ex/0301015].

[2] L. Huo., "In-Jet Tracking Efficiency Analysis for the STAR Time Projection Chamber in Polarized Proton-Proton Collisions at $\sqrt{s} = 200$ GeV", Master's thesis, Texas A&M University (2012)

[3] M. A. Lisa, "The STAR TPC Clusterfinder/Hitfinder", SN0238 (1996)

[4] G. Van Buren, L. Didenko, J. Dunlop, Y. Fisyak, J. Lauret, A. Lebedev, B. Stringfellow, J. Thomas and H. Wieman, Nucl. Instrum. Meth. A **566**, 22-25 (2006) doi:10.1016/j.nima.2006.05.131 [arXiv:physics/0512157 [physics]].

[5] F. Böhmer, M. Ball, S. Dorheim, C. Höppner, B. Ketzer, I. Konorov, S. Neubert, S. Paul, J. Rauch and M. Vandenbroucke, Nucl. Instrum. Meth. A **719**, 101-108 (2013) doi:10.1016/j.nima.2013.04.020 [arXiv:1209.0482 [physics.ins-det]].

[6] I. Chakaberia, "Grid leak simulation for STAR TPC using GARFIELD", SN0676 (2018)

[7] D. Smirnov, J. Lauret, V. Perevoztchikov, G. Van Buren and J. Webb, J. Phys. Conf. Ser. **898**, no. 4, 042058 (2017). doi:10.1088/1742-6596/898/4/042058

[8] C. Gagliardi, Jet Finding Techniques discussion forum, `https://www.star.bnl.gov/HyperNews-star/protected/get/jetfinding/992/1.html`

[9] D. Kalinkin, Jet Meeting (July 4, 2018), `https://drupal.star.bnl.gov/STAR/blog/veprbl/vpdmb-trigger-bias-jets`

[10] L. Adamczyk, L. Fulek, R. Sikora, "Supplementary note on diffractive analyses of 2015 proton-proton data", PSN0732 (2020)

[11] H. Matis, R. Bellwied, D. Irmscher, P. Jacobs, S. Klein, M. Lisa, I. Sakrejda, D. Weerasundara, P. Yepes, "STAR Geometry", SN0229 (1995)

A Embedding of single positrons

The single particle embedding study described in this note uses the samples with single positive pions embedded to make conclusions about tracking efficiency uncertainty. While we don't expect that the kind of the particle embedded will affect the study, it appears to be worthwhile to present a result for a different kind of particles produced with the same setup. For this purpose, the same samples were reproduced with single positrons embedded. The choice of positrons was motivated by desire for the largest possible difference in the mass scale to enhance possible material interaction effects. The pseudorandom number generator seeds were initialized with the same values as for the positive pion sample, this way the vertex distributions and momenta of the thrown particles are matching between the productions on the event-by-event basis.

The respective results for efficiency and cut pass rate for positrons are presented on figs. 54 and 55.

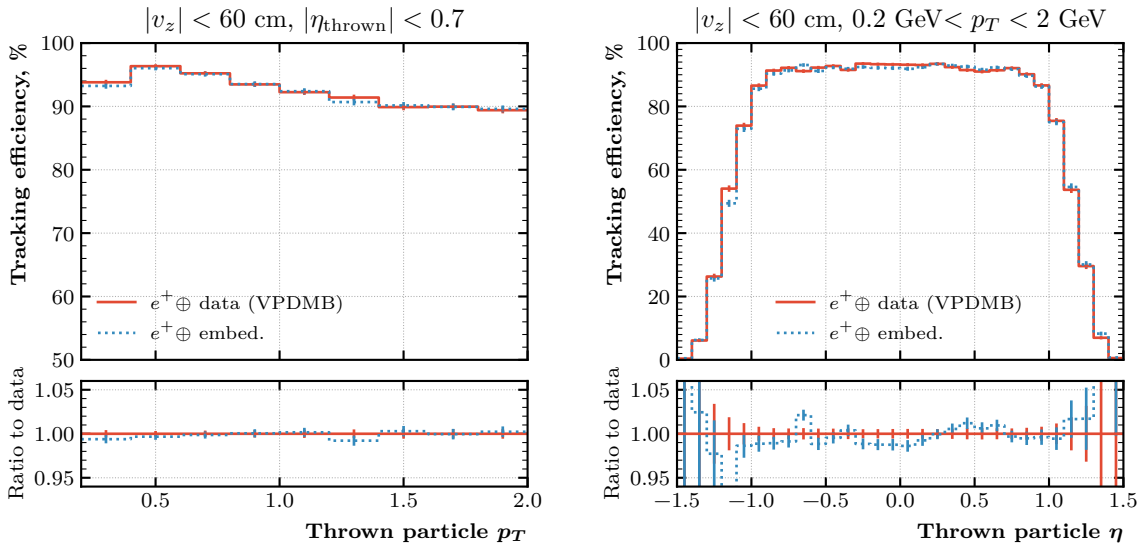

Figure 54: Tracking efficiency for positrons as a function of track p_T and track η in run 13059025 ($\sqrt{s} = 200$ GeV).

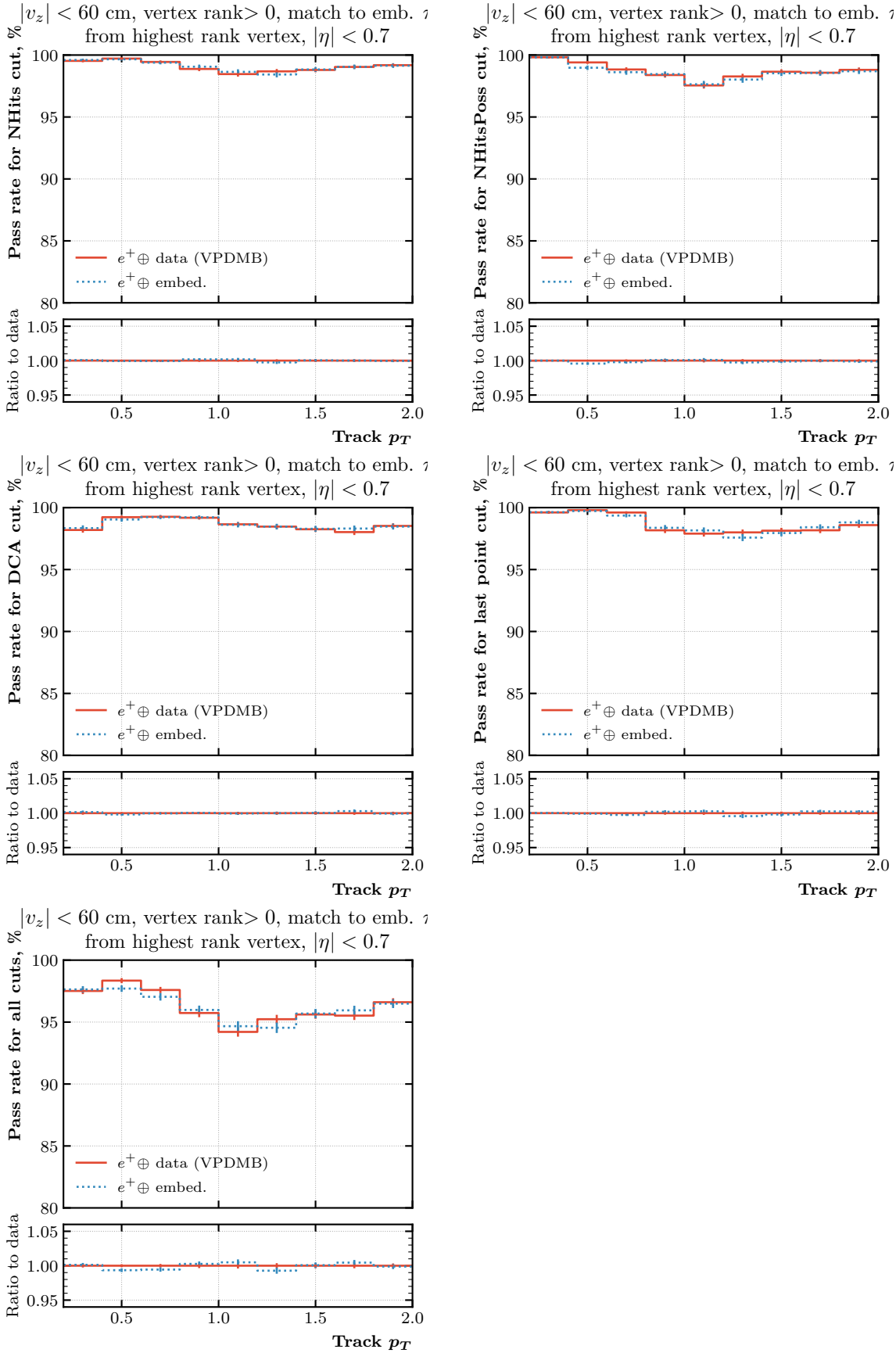

Figure 55: Fraction of tracks associated to the thrown positrons that pass a given track QA cut in run 13059025 ($\sqrt{s} = 200$ GeV).

B Single particle embedding code

In order to achieve the goals of this analysis the setups for single track embedding had to satisfy the following requirements:

- The setups should be able to optionally embed a single track. The productions without the single particles were used for production QA in section 4.3 and in data-driven studies in section 6. Those *have to* match the procedure for the respective official productions for data and embedding. The productions with the single particles were then used to determine the efficiencies in section 5.
- The origins of the single particles must be seeded consistently in embedding into data and embedding into embedding or embedding into pure MC. This was achieved by seeding the vertex positions from the vertices reconstructed in productions without the embedded particles.
- During the production the single tracks are to be mixed in at the ADC level for both data and embedding.

Let us start by considering a standard setup used for the embedding production in STAR. It employs two different frameworks:

1. A Fortran-based `starsim` generates events using Pythia 6 and propagates particles through the detector using GEANT 3. This step is steered with programs written in `*.kumac` files and output is writen to the "ZEBRA" `*.fzd` files.
2. An "embedding macro" written in C++ defines several "BFC"'s to read the simulated `*.fzd` files and the raw `*.daq` data files, perform event mixing, offline hit reconstruction, track reconstruction, vertex finding and produce `*.root` files with reconstructed events.

The analysis requirements create a need to perform a "double embedding" where the zero-bias data would be mixed with two simulations at once, so that one of the simulations would come from a normal Pythia event and the other would provide the additional single simulated particle thrown into the detector. Unfortunately there doesn't seem to be a straightforward way to read two `*.fzd` files in the same process. The simplest workaround of throwing an additional particle in the same `*.fzd` along with Pythia event violates at least one of the analysis requirements.

The approach taken for this analysis overcomes the `*.fzd` input file limit by passing fully simulated (GEANT+StTpcRs) tracks via simple ROOT trees of one `StTpcRawData` structure per event. This allows to achieve the goals of the analysis with minimal modification to the standard procedure.

The implementation starts with just two simple makers. The first one takes `StTpcRawData` of the current event and writes (`Sink`) it to a file:

StRoot/StTpcRawDataSink/StTpcRawDataSink.h

```
#pragma once

#include <string>

#include <TFile.h>
#include <TTree.h>

#include <StChain/StMaker.h>
#include <StEvent/StTpcRawData.h>

/**
 * Dumps StTpcRawData into a ROOT file
 */
class StTpcRawDataSink : public StMaker {
public:

  StTpcRawDataSink(const char *name = "tpc_raw_data_sink") : StMaker(name) {};
  virtual ~StTpcRawDataSink() {};
```

```
  Int_t Init();
  Int_t Make();
  Int_t Finish();

  void set_skip(bool flag);

  ClassDef(StTpcRawDataSink, 0);

private:

  TFile *_outfile;
  TTree *_tree;
  StTpcRawData *_tpc_raw_data;
  bool _enable_skip;
};
```

The maker can optionally take a second responsibility (when `set_skip(true)` is set) to terminate the processing of the current event. The purpose of this will be explained later. This implementation is straightforward:

StRoot/StTpcRawDataSink/StTpcRawDataSink.cxx

```
#include <type_traits>
#include <TObject.h>
#include <StEvent/StTpcRawData.h>
static_assert(
  std::is_base_of<TObject,StDigitalPair>::value,
  "This version of StEvent doesn't support serialization of StTpcRawData"
);
#include <TFile.h>
#include <StChain/StChainOpt.h>
#include <St_base/StMessMgr.h>

#include "StTpcRawDataSink.h"

Int_t StTpcRawDataSink::Init() {
  TString output_filename = GetChainOpt()->GetFileOut();
  output_filename.ReplaceAll(".root", ".tpcraw.root");
  _outfile = new TFile(output_filename.Data(), "RECREATE");
  _tree = new TTree("tpc_raw", "");
  _tpc_raw_data = nullptr;
  _tree->Branch("tpc_raw", &_tpc_raw_data);
  return kStOK;
}

Int_t StTpcRawDataSink::Make() {
  LOG_INFO << "Make()" << endm;

  Int_t retcode = kStOk;
  static StTpcRawData dummy;

  TObjectSet *event = dynamic_cast<TObjectSet*>(GetDataSet("Input"));
  if (!event) {
    LOG_ERROR << "StTpcRawDataSink: can't find input event" << endm;
    _tpc_raw_data = &dummy;
    retcode = kStWarn;
  }
  else
```

```
  {
    _tpc_raw_data = dynamic_cast<StTpcRawData*>(event->GetObject());
    if (!_tpc_raw_data) {
      LOG_ERROR << "StTpcRawDataSink: can't get StTpcRawData instance" << endm;
      _tpc_raw_data = &dummy;
      retcode = kStWarn;
    }
  }

  _tree->Fill();

  if (_enable_skip) {
    retcode = kStSkip;
  }

  return retcode;
}

Int_t StTpcRawDataSink::Finish() {
  auto _dir = gDirectory;
  _outfile->cd();
  _tree->Write();
  gDirectory = _dir;
  delete _tree; _tree = nullptr;
  delete _outfile; _outfile = nullptr;
  return kStOK;
}

/**
 * If StTpcRawDataSink is inserted in the midle of a big chain (like
 * embedding), then, oftentimes, we are not interested in further processing
 * the current event. This option makes StTpcRawDataSink automatically skip to
 * the next event after dumping StTpcRawData.
 */
void StTpcRawDataSink::set_skip(bool flag) {
  _enable_skip = flag;
  SetAttr(".Privilege", _enable_skip ? 1 : 0);
}
```

The second complementary maker reads (`Source`) the trees. It also has a second responsibility of mixing read `StTpcRawData` into the current event. The `ApplyTpcRawDataTruthIdOffset` method allows to add arbitrary offsets to the truth id numbers of the embedded signals. The definition and implmenetation are the following:

StRoot/StTpcRawDataSourceMixer/StTpcRawDataSourceMixer.h

```
#pragma once

#include <string>

#include <TFile.h>
#include <TTree.h>

#include <StChain/StMaker.h>
#include <StEvent/StTpcRawData.h>

/**
 * Read StTpcRawData from a ROOT, tree then mix it into TpxRaw event
```

```
 */
class StTpcRawDataSourceMixer : public StMaker {
public:

  StTpcRawDataSourceMixer(std::string &filename)
    : id_offset(0)
    , _filename(filename)
  {}
  virtual ~StTpcRawDataSourceMixer() {};

  Int_t Init();
  Int_t Make();
  Int_t Finish();

  static void ApplyTpcRawDataTruthIdOffset(StTpcRawData *trd, UShort_t id_offset);
  UShort_t id_offset;

  ClassDef(StTpcRawDataSourceMixer, 0);

private:

  std::string _filename;
  TFile *_infile;
  TTree *_tree;
  Long64_t _entry;
  StTpcRawData *_tpc_raw_data;
};
```

StRoot/StTpcRawDataSourceMixer/StTpcRawDataSourceMixer.cxx

```
#include <type_traits>
#include <TObject.h>
#include <StEvent/StTpcRawData.h>
static_assert(
  std::is_base_of<TObject,StDigitalPair>::value,
  "This version of StEvent doesn't support serialization of StTpcRawData"
);

#include "StTpcRawDataSourceMixer.h"

Int_t StTpcRawDataSourceMixer::Init() {
  _infile = new TFile(_filename.c_str());
  _tree = dynamic_cast<TTree*>(_infile->Get("tpc_raw"));
  if (!_tree) {
    LOG_ERROR << "StTpcRawDataSourceMixer: can't get input tree" << endm;
    return kStErr;
  }
  _entry = 0;
  _tpc_raw_data = nullptr;
  _tree->SetBranchAddress("tpc_raw", &_tpc_raw_data);
  return kStOK;
}

void StTpcRawDataSourceMixer::ApplyTpcRawDataTruthIdOffset(
  StTpcRawData *trd, UShort_t _id_offset
) {
  UInt_t num_sectors = trd->getNoSectors();
```

```
  for (UInt_t sector_id = 1; sector_id <= num_sectors; sector_id++) {
    StTpcDigitalSector *sector = trd->getSector(sector_id);
    if (!sector) continue;
    StDigitalSector &sector_rows = *sector->rows();

    for (StDigitalSector::iterator row_it = sector_rows.begin();
         row_it != sector_rows.end(); ++row_it) {
      StDigitalPadRow &row = *row_it;

      for (StDigitalPadRow::iterator pad_it = row.begin(); pad_it != row.end();
           ++pad_it) {
        StDigitalTimeBins &pad = *pad_it;

        for (StDigitalTimeBins::iterator time_bin_it = pad.begin();
             time_bin_it != pad.end(); ++time_bin_it) {
          StDigitalPair &time_bin = *time_bin_it;

          if (!time_bin.isIdt()) continue;
          UShort_t *idt = time_bin.idt();
          assert(idt);
          Int_t size = time_bin.size();
          for (Int_t ix = 0; ix < size; ix++) {
            idt[ix] += _id_offset;
          }
        }
      }
    }
  }
}

Int_t StTpcRawDataSourceMixer::Make() {
  LOG_INFO << "StTpcRawDataSourceMixer::Make()" << endm;

  TObjectSet *event = dynamic_cast<TObjectSet*>(GetDataSet("Output"));
  if (!event) {
    LOG_ERROR << "StTpcRawDataSourceMixer: can't find output event" << endm;
    return kStErr;
  }
  StTpcRawData *daq_raw_data = dynamic_cast<StTpcRawData*>(event->GetObject());
  if (!daq_raw_data) {
    LOG_ERROR
      << "StTpcRawDataSourceMixer: can't get StTpcRawData instance"
      << endm;
    return kStErr;
  }

  Int_t read = _tree->GetEntry(_entry);
  if (read <= 0) {
    LOG_ERROR << "StTpcRawDataSourceMixer: can't read input tree" << endm;
    return kStErr;
  }

  ApplyTpcRawDataTruthIdOffset(_tpc_raw_data, id_offset);
  *daq_raw_data += *_tpc_raw_data;

  _entry++;
```

```
  return kStOK;
}

Int_t StTpcRawDataSourceMixer::Finish() {
  delete _tpc_raw_data; _tpc_raw_data = nullptr;
  delete _tree; _tree = nullptr;
  delete _infile; _infile = nullptr;
  return kStOK;
}
```

We first introduce the production for the data as it is the simpler one. It consists of a macro that by default only setups a single BFC with the options as described in section 4.1.

macros/data_mixer.C

```
void data_mixer(
  unsigned int num_events,
  const char *bfc_options,
  const char *daq_filename,
  const char *tpc1_input = NULL,
  const char *tpc2_input = NULL,
  const char *output_filename="output.root"
  )
{
  gROOT->LoadMacro("bfc.C");
  if (gClassTable->GetID("StBFChain") < 0) Load();
  gSystem->Load("StBichsel");
  gSystem->Load("StEvent");
  gSystem->Load("StTpcRawDataSourceMixer");
  gSystem->Load("StSetTruthIdMaker");

  StBFChain* chain0 = 0;
  bfc(-1, bfc_options, daq_filename, output_filename);
  chain0 = chain;
  StTpcRawDataSourceMixer *s1, *s2;
  if (tpc1_input) {
    s1 = new StTpcRawDataSourceMixer(tpc1_input);
    s1->SetName("tpc_raw_data_sink1");
    s1->SetOutput("Output", "TpxRaw/.data/Event");
    s1->id_offset = 500;
  }
  if (tpc2_input) {
    s2 = new StTpcRawDataSourceMixer(tpc2_input);
    s2->SetName("tpc_raw_data_sink2");
    s2->SetOutput("Output", "TpxRaw/.data/Event");
    chain0->AddAfter("TpxRaw", s2);
  }
  if (tpc1_input) {
    chain0->AddAfter("TpxRaw", s1);
  }
  StSetTruthIdMaker *truth_maker = new StSetTruthIdMaker;
  chain0->AddBefore("MuDst", truth_maker);

  StMaker::lsMakers(chain0);
  chain0->Init();
  chain0->EventLoop(num_events);
  chain0->Finish();
}
```

This implementation has up to two `StTpcRawDataSourceMixer`'s available for setup, but in practice only one was used.

One can also notice an additional `StSetTruthIdMaker` maker. It's purpose is to implement a missing call to the `StEvent::setIdTruth()` which is done in embedding by some other maker, but is also needed here to assign the truth id numbers to the reconstructed tracks. It is implemented as:

StRoot/StSetTruthIdMaker/StSetTruthIdMaker.h

```
#pragma once

#include <StChain/StMaker.h>

/**
 * In standard embedding StEvent::setIdTruth() is called by
 * StMuDstMaker::fillMC(), but latter requires geant datasets to be
 * present. This simple maker will do the job instead.
 */
class StSetTruthIdMaker : public StMaker {
public:

  virtual ~StSetTruthIdMaker() {};

  Int_t Make();

  ClassDef(StSetTruthIdMaker, 0);
};
```

StRoot/StSetTruthIdMaker/StSetTruthIdMaker.cxx

```
#include <St_base/StMessMgr.h>
#include <StEvent/StEvent.h>

#include "StSetTruthIdMaker.h"

Int_t StSetTruthIdMaker::Make() {
  LOG_INFO << "Calling StEvent::setIdTruth()" << endm;
  StEvent *st_event = dynamic_cast<StEvent*>(GetInputDS("StEvent"));
  st_event->setIdTruth();
  return kStOk;
}
```

The embedding macro is based on the official macros used in run12 pp200 and pp510 embedding. As before, additional makers were inserted in strategic points. Depending on the configuration, this macro can serve as a macro that performs double embedding of one `*.daq`, one `*.fzd` and one single track from the `StTpcRawData` ROOT file. The other more exotic configuration employs reading `*.daq` files to set the timestamps and ensure the loading of all the needed database constants, for `*.fzd` file, a file with a single particle is used, and, finally, `StTpcRawDataSink` writes fully simulated `StTpcRawData` events to a file. The event processing is then interrupted using `set_skip(true)` to not perform mixing, offline hit reconstruction, track reconstruction and vertex finding. This is done as an optimization.

macros/mixer.C

```
////////////////////////////////////////////////////////////////////////
//
// $Id: bfcMixer_pyhia.C,v 1.0 2014/07/08 23:09:54 Jinlong Zhang $
//
// $combine bfcMixer_Jet.C and bfcMixer_W.C into bfcMixer_pythia.C
//
```

```
// $Log: bfcMixer_Jet.C,v $
// Revision 1.1  2013/02/10 23:09:54  pibero
// Simulations
//
//
// JET EMBEDDING MACRO
//
// Pibero Djawotho <pibero@tamu.edu>
// Texas A&M University
// 27 July 2011
//
////////////////////////////////////////////////////////////////////////////

class StChain;
StChain* Chain = 0;

class StBFChain;
StBFChain* chain1 = 0;
StBFChain* chain2 = 0;
StBFChain* chain3 = 0;

//_____________________________________________________________________
void mixer(const Int_t Nevents = 1000,
                  const Char_t* daqfile = "@run10148002.list",
                  const Char_t* fzdfile =
                  "eliza14/SL11d_embed/10148002/pt11_15_10148002_1.fzd",
                  bool sink_mode = false,
                  const Char_t* extra_mixer_input = NULL,
                  const Char_t* prodName = "P11idpp200RFF",
                  const Char_t* flag = "Jet",
                  bool trgfilter = 0,
                  bool do_wipe_tpcrawdata = false )
{
  // Run 12 Collins, Kevin's request  // I add "sti and remove AgML to fix errors
  TString prodP12idpp200("DbV20130212 DbV20160506_EMC_Calibrations pp2012b "
                         "Sti mtdDat btof fmsDat VFPPVnoCTB useBTOF4Vtx "
                         "beamline BEmcChkStat Corr4 OSpaceZ2 OGridLeak3D "
                         "-hitfilt -evout");
  //Run12 pp500 RFF chain
   TString prodP13ibpp500RFF("DbV20130502 DbV20160318_EMC_Calibrations "
                             "Dbv20160318_TRG_Calibrations pp2012b Sti AgML "
                             "mtdDat btof fmsDat VFPPVnoCTB beamline BEmcChkStat "
                             "Corr4 OSpaceZ2 OGridLeak3D "
                             "-hitfilt -evout");

  // Additional tags needed for embedding
  prodP12idpp200 += " TpxClu -VFMinuit VFPPVnoCTB beamLine -hitfilt";
  prodP13ibpp500RFF += " TpxClu -VFMinuit";

  TString geomP12id("ry2012a");
  TString geomP13ib("ry2012a");
  TString chain1Opt("in,magF,tpcDb,NoDefault,TpxRaw,-ittf,NoOutput");
  TString chain2Opt("fzin,gen_T,geomT,sim_T,TpcRS,-ittf,-tpc_daq,nodefault");
  chain2Opt += " ";

  TString chain3Opt;
  if (prodName == "P12idpp200")
```

```
{ chain3Opt = prodP12idpp200;    chain2Opt += geomP12id; }
else if (prodName == "P13ibpp500RFF")
{ chain3Opt = prodP13ibpp500RFF; chain2Opt += geomP13ib; }
else {
  cout << "Choice prodName " << prodName
       << " does not correspond to known chain. Processing impossible."
       << endl;
  return;
}
chain3Opt.Prepend(' ');
//chain3Opt.Prepend(DbVoption);
chain3Opt += ",Embedding,TpcMixer,GeantOut,MiniMcMk,McAna,"
             "-in,NoInput,useInTracker,nodefault";
chain3Opt += ",";

if (prodName == "P12idpp200"){ chain3Opt += geomP12id; } // JLZhang.
else if (prodName == "P13ibpp500RFF"){ chain3Opt += geomP13ib; }
else {
  cout << "Choice prodName " << prodName
       << " does not correspond to known chain. Processing impossible. "
       << endl;
  return;
}

// Add BEMC simulators to chain
chain3Opt += ",emcSim";

// Add EEMC fast simulator to chain
chain3Opt += ",EEfs";

// Dynamically link some shared libs
gROOT->LoadMacro("bfc.C");
if (gClassTable->GetID("StBFChain") < 0) Load();
gSystem->Load("StBichsel");
gSystem->Load("StEvent");
if (sink_mode) {
  gSystem->Load("StTpcRawDataSink");
}
if (extra_mixer_input) {
  gSystem->Load("StTpcRawDataSourceMixer");
}
if (do_wipe_tpcrawdata) {
  gSystem->Load("StTpcRawDataWipeMaker");
}
//______________Create the main chain object_____________________________________
Chain = new StChain("Embedding");
//______________________________________________________________________________
bfc(-1,chain1Opt,daqfile);
chain1 = chain;
chain1->SetName("One");
// Use DB cache to reduce overhead
chain1->SetAttr(".call","SetActive(0)","St_db_Maker::");
Chain->cd();
//______________________________________________________________________________
bfc(-1,chain2Opt,fzdfile);
chain2 = chain;
chain2->SetName("Two");
```

```
Chain->cd();
if (chain2->GetOption("TRS")){
  StTrsMaker *trsMk = (StTrsMaker *) chain2->GetMaker("Trs");
  if (!trsMk) {
    cout << "Cannot find Trs in chain2" << endl;
    return;
  }
  trsMk->setNormalFactor(1.32);
  trsMk->SetMode(0);
}
//_____________________________________________________________________________
// gSystem->Load("StFtpcMixerMaker");
// StFtpcMixerMaker  *ftpcmixer = new StFtpcMixerMaker("FtpcMixer","daq","trs");
//_____________________________________________________________________________
TString OutputFileName(gSystem->BaseName(fzdfile));
OutputFileName.ReplaceAll("*","");
OutputFileName.ReplaceAll(".fzd","");
//  OutputFileName.Append("_emb.root");
OutputFileName.Append(".root");
bfc(-1,chain3Opt,0,OutputFileName);
chain3 = chain;
chain3->SetName("Three");
Chain->cd();
//_____________________________________________________________________________
StTpcMixerMaker  *mixer = (StTpcMixerMaker *) chain3->Maker("TpcMixer");
mixer->SetInput("Input1","TpxRaw/.data/Event");

if (chain2Opt.Contains("TpcRS",TString::kIgnoreCase)) {
  mixer->SetInput("Input2","TpcRS/Event");
} else {
  mixer->SetInput("Input2","Trs/.const/Event");
}
Chain->cd();

if(flag=="W"){
// blacklist some detectors to save DB load
St_db_Maker* dbMk = (St_db_Maker*)chain3->GetMaker("db");
dbMk->SetAttr("blacklist", "svt");
dbMk->SetAttr("blacklist", "ssd");
//  dbMk->SetAttr("blacklist", "ftpc"); // S.F.
}

//-------------------------------- EMC MIXERS ---------------------------------
// Add BEMC mixer to chain3
StEmcRawMaker* emcRaw = (StEmcRawMaker*)chain3->GetMaker("emcRaw");
emcRaw->getBemcRaw()->saveAllStEvent(true); // use all 4800 BEMC towers
gSystem->Load("StEmcMixerMaker");
StEmcMixerMaker* bemcMixer = new StEmcMixerMaker;
chain3->AddAfter("EmcSimulator",bemcMixer);
// Set EEMC fast and slow simulator in embedding mode
StEEmcFastMaker* eefs = (StEEmcFastMaker*)chain3->GetMaker("eefs");
eefs->SetEmbeddingMode(); // Use local StEmcCollection
eefs->UseFullTower(true); // Use full ETOW detector
StEEmcSlowMaker* eess = new StEEmcSlowMaker;
eess->setEmbeddingMode(true);
// Add EEMC mixer to chain3
StEEmcMixerMaker* eemcMixer = new StEEmcMixerMaker;
```

```
//-----------------------------------------------------------------------------

//----------------------------- TRIGGER FILTER ------------------------------
// We want to achieve the following ordering for makers:
// 1. BBC simulator
// 2. BEMC simulator
// 3. BEMC mixer
// 4. EEMC fast simulator
// 5. EEMC slow simulator
// 6. EEMC mixer
// 7. Pythia event maker
// 8. Trigger simulator
// 9. Trigger filter
// 10. TPC maker

// Place TPC chain after EMC makers
chain3->AddAfter("eefs",chain3->GetMaker("tpcChain"));
chain3->AddAfter("eefs",eemcMixer);
chain3->AddAfter("eefs",eess);

if(flag=="Jet"){
// Place Pythia maker after GEANT maker
// and trigger filter after EMC makers
gSystem->Load("StJetSkimEvent");
gSystem->Load("StMCAsymMaker");
gSystem->Load("StBfcTriggerFilterMaker");

StPythiaEventMaker* pythia = new StPythiaEventMaker;
TString pyfile = gSystem->BaseName(fzdfile);
pyfile.ReplaceAll(".fzd",".pythia.root");
pythia->SetPythiaFile(pyfile);
chain3->AddAfter("geant",pythia);

// Place trigger simulator after EMC makers
gSystem->Load("StTriggerUtilities");
StTriggerSimuMaker* trgsim = new StTriggerSimuMaker;
//trgsim->useOnlineDB();
trgsim->useOfflineDB();
trgsim->setMC(1);
// BBC was not used in Run 9 jet triggers
//trgsim->useBbc();
//trgsim->bbc->setSource("StEvent");
trgsim->useBemc();
trgsim->bemc->setConfig(StBemcTriggerSimu::kOnline);
trgsim->useEemc();
trgsim->eemc->setSource("StEvent");

if (prodName == "P12idpp200") {
/*
trgsim->bemc->setBarrelJetPatchTh0(32);
trgsim->bemc->setBarrelJetPatchTh1(43);
trgsim->bemc->setBarrelJetPatchTh2(64);
*/
// set for Run-11 based on trigger versioning page 20130728
trgsim->setBarrelJetPatchTh(0,32);
trgsim->setBarrelJetPatchTh(1,43);
trgsim->setBarrelJetPatchTh(2,64);
```

```
trgsim->setOverlapJetPatchTh(0,32);
trgsim->setOverlapJetPatchTh(1,43);
trgsim->setOverlapJetPatchTh(2,64);

trgsim->setEndcapJetPatchTh(0,32);
trgsim->setEndcapJetPatchTh(1,43);
trgsim->setEndcapJetPatchTh(2,64);

trgsim->setBarrelHighTowerTh(0,11);
trgsim->setBarrelHighTowerTh(1,18);
trgsim->setBarrelHighTowerTh(2,25);
trgsim->setBarrelHighTowerTh(3,31);

trgsim->setEndcapHighTowerTh(0,25);
trgsim->setEndcapHighTowerTh(1,31);
}


if(trgfilter){
      StBfcTriggerFilterMaker* trgfilt = new StBfcTriggerFilterMaker;
      // no trigger filter for Run-11 VPDMB
      trgfilt->SetOkAllEvents(1);
      //The BFC trigger filter will select only JP1, AJP and BHT3 events
      trgfilt->SetJP1();
      trgfilt->SetAJP();
      trgfilt->SetBHT3();

      // Lower all jet patch thresholds by one unit from
      // their values obtained from the database using
      // the current timestamp.
      trgfilt->changeJPThresh(-1);
      // no trigger filter for Run-11 VPDMB
      chain3->AddBefore("tpcChain",trgfilt);
}

chain3->AddBefore("tpcChain",trgsim);

// Move these makers after trigger decision
// *** VERY IMPORTANT ***
// The order of TpxRaw and TpcRS *must* be preserved
// or the embedding will *not* work. [RT# 2299]
// http://www.star.bnl.gov/rt2/Ticket/Display.html?id=2299
StTpcRSMaker* TpcRS = (StTpcRSMaker*)chain2->GetMaker("TpcRS");
StTpcHitMaker* TpxRaw = (StTpcHitMaker*)chain1->GetMaker("TpxRaw");
chain3->AddBefore("TpcMixer",TpxRaw);
chain3->AddBefore("TpcMixer",TpcRS);
if (sink_mode) {
  chain3->cd(); // needed for GetChainOpt() to work
  StTpcRawDataSink *sink = new StTpcRawDataSink;
  sink->SetInput("Input", "TpcRS/Event");
  sink->set_skip(true);
  chain3->AddBefore("TpcMixer", sink);
  Chain->cd();
}

if (do_wipe_tpcrawdata) {
```

```
    StTpcRawDataWipeMaker *tpcrawdatareset = new StTpcRawDataWipeMaker;
    tpcrawdatareset->SetOutput("Target", "TpxRaw/.data/Event");
    chain3->AddBefore("TpcMixer", tpcrawdatareset);
  }

  if (extra_mixer_input) {
    StTpcRawDataSourceMixer *source =
      new StTpcRawDataSourceMixer(extra_mixer_input);
    source->SetName("tpc_raw_data_mixer");
    source->SetOutput("Output", "TpxRaw/.data/Event");
    source->id_offset = 500;
    //chain3->AddAfter("tpc_raw_data_sink", source);
    chain3->AddBefore("TpcMixer", source);
  }

#if 0
  // Turn on debugging of DB maker
  St_db_Maker* db = (St_db_Maker*)chain1->GetMaker("db");
  db->SetDebug(2);
#endif

  //-----------------------------------------------------------------------
  TString trgfile = gSystem->BaseName(fzdfile);
  trgfile.ReplaceAll(".fzd",".trig.root");
  TFile* ofile = TFile::Open(trgfile,"recreate");
  assert(ofile);
  TH2F* hBarrelHighTowerSimu = new TH2F("hBarrelHighTowerSimu",
    "BEMC high tower simu;trigger patch;high tower",300,0,300,64,0,64);
  TH2F* hBarrelPatchSumSimu = new TH2F("hBarrelPatchSumSimu",
    "BEMC patch sum simu;trigger patch;patch sum",300,0,300,64,0,64);
  TH2F* hEndcapHighTowerSimu = new TH2F("hEndcapHighTowerSimu",
    "EEMC high tower simu;trigger patch;high tower",90,0,90,64,0,64);
  TH2F* hEndcapPatchSumSimu = new TH2F("hEndcapPatchSumSimu",
    "EEMC patch sum simu;trigger patch;patch sum",90,0,90,64,0,64);
  TH2F* hBarrelJetPatchSimu = new TH2F("hBarrelJetPatchSimu",
    "BEMC jet patch;jet patch;adc",18,0,18,160,0,160);
  TH2F* hEndcapJetPatchSimu = new TH2F("hEndcapJetPatchSimu",
    "EEMC jet patch;jet patch;adc",6,0,6,160,0,160);
  TH2F* hOverlapJetPatchSimu = new TH2F("hOverlapJetPatchSimu",
    "BEMC-EEMC-overlap;jet patch;adc",6,0,6,160,0,160);
  }
  //-----------------------------------------------------------------------
  // Initialize chain
  cout<<"debug0++++++++++++++++++++++++"<<endl;
  Chain->Init();
  StMaker *geant_maker = Chain->GetMakerInheritsFrom("St_geant_Maker");
  geant_maker->SetAttr("KeepRunNumber", 1);
  geant_maker->SetAttr("Don'tTouchTimeStamp", 1);
  cout<<"debug1++++++++++++++++++++++++"<<endl;
  PrintTimer(Chain);
  puts("Order of makers in BFCMIXER:");
  StMaker::lsMakers(Chain);
  // Event loop
  int mNTotal = 0;
  int mNFailed = 0;
  TBenchmark evnt;
  StIOMaker* inputStream = (StIOMaker*)chain1->GetMaker("inputStream");
```

```
  for (int iEvent = 1; iEvent <= Nevents; ++iEvent) {
    evnt.Reset();
    evnt.Start("QAInfo:");
    Chain->Clear();
    int iMake = Chain->Make(iEvent);
    if (iMake == kStErr) ++mNFailed;
    if (inputStream->GetMakeReturn() % 10 == kStEOF) {
      inputStream->Rewind();
      --iEvent;
      continue;
    }
    //if (iMake == kStSkip) continue;
    if (iMake % 10 == kStEOF || iMake % 10 == kStFatal) break;
    ++mNTotal;
    PrintTimer(Chain);
    //--------------------------------------------------------------------------

        if(flag=="Jet") {
    // BEMC high towers and trigger patches
    for (int triggerpatch = 0; triggerpatch < 300; ++triggerpatch) {
      hBarrelHighTowerSimu->Fill(triggerpatch,
        trgsim->bemc->getBEMC_FEE_HT_ADC()[triggerpatch]);
      hBarrelPatchSumSimu->Fill(triggerpatch,
        trgsim->bemc->getBEMC_FEE_TP_ADC()[triggerpatch]);
    } // for triggerpatch
    // BEMC jet patches
    for (int jetpatch = 0; jetpatch < 18; ++jetpatch) {
      hBarrelJetPatchSimu->Fill(jetpatch,
        trgsim->bemc->barrelJetPatchAdc(jetpatch));
    } // for jetpatch
    // EEMC high towers and trigger patches
    for (int triggerpatch = 0; triggerpatch < 90; ++triggerpatch) {
      hEndcapHighTowerSimu->Fill(triggerpatch,
        trgsim->eemc->getEndcapHighTower(triggerpatch));
      hEndcapPatchSumSimu->Fill(triggerpatch,
        trgsim->eemc->getEndcapPatchSum(triggerpatch));
    } // for triggerpatch
    // EEMC jet patches
    for (int jetpatch = 0; jetpatch < 6; ++jetpatch) {
      hEndcapJetPatchSimu->Fill(jetpatch,
        trgsim->eemc->endcapJetPatchAdc(jetpatch));
    } // for jetpatch
    // BEMC-EEMC-overlap jet patches
    for (int i = 0; i < 2; ++i) {
      int jetpatch, adc;
      trgsim->emc->getOverlapJetPatchAdc(i,jetpatch,adc);
      hOverlapJetPatchSimu->Fill(jetpatch,adc);
    } // for i
        }

    //--------------------------------------------------------------------------
    evnt.Stop("QAInfo:");
    printf("QAInfo: Done with Event [no. %d/run %d/evt. %d/Date.Time %d.%d/sta %d]"
           " Real Time = %10.2f seconds Cpu Time = %10.2f seconds\n",
           iEvent,Chain->GetRunNumber(),Chain->GetEventNumber(),Chain->GetDate(),
           Chain->GetTime(),chain3->GetMakeReturn(),evnt.GetRealTime("QAInfo:"),
           evnt.GetCpuTime("QAInfo:"));
```

```
  } // End event loop
  printf("QAInfo:EventLoop completed code %d\n",iMake);
  gSystem->Exec("date");
  TDatime t;
  printf("QAInfo:Run is finished at Date/Time %i/%i; Total events processed: %i"
         " and not completed: %i\n",t.GetDate(),t.GetTime(),mNTotal,mNFailed);
  //-----------------------------------------------------------------------
  ofile->Write();
  ofile->Close();
  //-----------------------------------------------------------------------
}

// Print timers for all makers in chain
class StMaker;
void PrintTimer(StMaker* chain)
{
  TIter next(chain->GetMakeList());
  StMaker* maker;
  while (maker = (StMaker*)next()) {
    maker->PrintTimer();
    PrintTimer(maker);
    // Hack to reset timer
    maker->StartTimer(true);
    maker->StopTimer();
  }
}
```

The final piece is another maker used to produce the "pure MC" simulation. This is done by configuring `mixer.C` to include a `StTpcRawDataWipeMaker` which wipes the TPC responses read from the DAQ file, so that the TPC hits are only coming from the simulation.

StRoot/StTpcRawDataWipeMaker/StTpcRawDataWipeMaker.h

```
#pragma once

#include <StChain/StMaker.h>

/**
 * Clears StTpcRawData in the event
 */
class StTpcRawDataWipeMaker : public StMaker {
public:

  StTpcRawDataWipeMaker(const char *name = "tpc_raw_data_wipe")
    : StMaker(name)
  {};
  virtual ~StTpcRawDataWipeMaker() {};

  Int_t Make();

  ClassDef(StTpcRawDataWipeMaker, 0);
};
```

StRoot/StTpcRawDataWipeMaker/StTpcRawDataWipeMaker.cxx

```
#include <StEvent/StTpcRawData.h>
#include <St_base/StMessMgr.h>
```

```
#include "StTpcRawDataWipeMaker.h"

Int_t StTpcRawDataWipeMaker::Make() {
  LOG_INFO << "Make()" << endm;

  TObjectSet *event = dynamic_cast<TObjectSet*>(GetDataSet("Target"));
  if (!event) {
    LOG_FATAL << "Can't find output event" << endm;
    return kStErr;
  }

  StTpcRawData *tpcrawdata = dynamic_cast<StTpcRawData*>(event->GetObject());
  if (!tpcrawdata) {
    LOG_FATAL << "Missing StTpcRawData object" << endm;
    return kStErr;
  }

  tpcrawdata->Clear();

  return kStOK;
}
```