Flexible Levy-Based Models for Time Series of Count Data with Zero-Inflation, Overdispersion, and Heavy Tails.

Confort Kollie [a,d], Philip Ngare [b] and Bonface Malenje [c]

Pan African University Institute for Basic Sciences, Technology and Innovation [a]
School of Mathematics, University of Nairobi [b]
Jomo Kenyatta University of Agriculture and Technology, Department of Statistics and Actuarial Sciences [c]
Corresponding author: confort.tina@students.jkuat.ac.ke
https://orcid.org/0009-0000-3701-9168

Abstract

This supplementary file provides detailed information on the formulas and R code in the main paper. It includes a description of the data generating process, and the R code used to implement the simulations. This information is provided to allow other researchers to replicate the simulations and to use the same formulas and R code to conduct their own simulations.

The supplementary file is divided into two sections. The first section provides the derivations under exponential and Sup-IG cases. The second section provides the R code for simulation and fitting data.

Keywords: formulas, R code

November 16, 2023

Nairobi, Kenya

Detailed computations:

We provide more detailed computations for the exponential case.

Let $\eta(x) = \exp(\lambda x)$, with $\lambda > 0$.

We note that $leb(D) = leb(D_0) = \int_{-\infty}^{0} \eta(s)\eta s = \int_{-\infty}^{0} \exp(\lambda s)\eta s = \frac{1}{\lambda} < \infty$.

Then, $\eta(s-t) = \exp(-\lambda(t-s)) = \exp(\lambda(s-t))$, where the autocorrelation function is given by $r(h) = Corr(Y_t, Y_{t+h}) = \exp(-\lambda h)$.

We provide more detailed computations for the Sup-IG case.

Let

$$\eta(t) = \left(1 - \frac{2t}{\gamma^2}\right)^{-1/2} \exp\left(\delta\gamma\left(1 - \sqrt{1 - \frac{2t}{\gamma^2}}\right)\right),$$

implying that,

$$\text{Leb}(D) = \int_0^\infty \eta(-t)\eta t = \int_0^\infty \left(1 - \frac{2t}{\gamma^2}\right)^{-1/2} \exp\left(\delta\gamma\left(1 - \sqrt{1 - \frac{2t}{\gamma^2}}\right)\right)\eta t$$

After substituting the variable $y = \sqrt{1 + \frac{2t}{\gamma^2}}$, we obtain the following:

$$\begin{aligned}
\text{Leb}(D) = \int_0^\infty \eta(-t)\eta t &= \int_1^\infty \frac{1}{y}\exp(\delta\gamma(1-y))\gamma^2 y\eta y \\
&= \gamma^2 \int_1^\infty \exp(\delta\gamma(1-y))\eta y \\
&= \gamma^2 \exp^{\delta\gamma} \int_1^\infty e^{-\delta\gamma y} \\
&= \frac{\gamma}{\delta}
\end{aligned}$$

Using the same method , we can define $\psi := \sqrt{\frac{2h}{\gamma} + 1}$ and derive the following equations:

$$\begin{aligned}
\text{Leb}(D_h \cap D) &= \int_h^\infty \eta(-t)\eta t \\
&= \gamma^2 \exp^{\delta\gamma} \int_\psi^\infty e^{-\delta\gamma y}\eta y \\
&= \frac{\gamma}{\delta} e^{\delta\gamma(1-\psi)}.
\end{aligned}$$

see e.g Barndorff-Nielsen et al. (2012) and Leonte and Veraart (2022).

R codes

```
library(ambit)
library(gamlss.dist)

spois<-function(n,L){

fp <- function(x, L) {
((L^(x + 2) * (x + 1)) / (factorial(x) * (exp(L) * (L^2 - L + 1) - 1) * (x + 2)
}
valuesp<-0:100
probp<-fp(valuesp,L)
CDFp<-cumsum(probp)
N<-n
genV<-rep(0,N)
for (i in 1:N){
U<-runif(1)
index<-min(which(CDFp>U))
genV[i]<-valuesp[index]
}
return(genV)
}

ComputeSliceSizes <- function(n, Delta, trawlfct){

#Define b_k=\int_0^{\Delta_n}a(k\Delta_n-x)dx, for k=1,..,n+1
b_vector <- numeric(n+1)
for(k in 1:(n+1)){
g_b <- function(x){trawlfct(x-k*Delta)}
b_vector[k] <- stats::integrate(g_b, 0, Delta)$value
}
```

```
#Define c_k=b_k-b_{k+1} for k=1,..,n
c_vector <- numeric(n)
for(k in 1:n){
c_vector[k] <- b_vector[k]-b_vector[k+1]
}
###For first column of slice matrix integrate from -infty rather than from 0
d_vector <- numeric(n+1)
for(k in 0:n){
g_d <- function(x){trawlfct(x-k*Delta)}
d_vector[k+1] <- stats::integrate(g_d, -Inf, 0)$value
}

#Define e_k=d_k-d_{k+1} for k=1,..,n
e_vector <- numeric(n)
for(k in 1:n){
e_vector[k] <- d_vector[k]-d_vector[k+1]
}

#####Compute matrix of slice sizes
slice_size_matrix <- matrix(0, n+1, n+1)

for(k in 1:n){
slice_size_matrix[k, 1:(n+1-k)] <- rep(c_vector[k], n+1-k)
slice_size_matrix[k,(n+1-k+1)] <- b_vector[k]
}
#Column of first slices:
slice_size_matrix[,1] <- c(e_vector, d_vector[n+1])

return(slice_size_matrix)
#return(list(b_vector, c_vector, d_vector, e_vector, slice_size_matrix))
```

```
}

#####
#Adding up the slices for a given (n+1)x(n+1) input matrix
#If the slicematrix contains the Lebesgue measure of the slices, then it
#returns the sequence of trawl sets.
#If the simulated slices are the input, then it returns a trawl path.
#
# #'@title AddSlices
# #'@param slicematrix A matrix of slices.
# #'@return Returns the sum of all slices
# #'@export

AddSlices <- function(slicematrix){
n <- nrow(slicematrix)-1
x <- numeric(n+1)
tmp <- 0
for(k in 0:n){
tmp<-0
for(j in 1:(k+1)){
tmp<-tmp+sum(slicematrix[(k+2-j):(n+2-j),j])
}
x[1+k]<-tmp
}
return(x)
}
#####
#Adding up the slices for a given (n+1)x(n+1) input matrix
#weighted by a kernel function
```

```
#If the simulated slices are the input, then it returns a path
#of an ambit process.
#
# #'@title AddWeightedSlices
# #'@param slicematrix A matrix of slices
# #'@param weightvector A vector of weights
# #'@return Returns the weighted sum of all slices
# #'@export

AddWeightedSlices <- function(slicematrix, weightvector){
n <- nrow(slicematrix)-1
x <- numeric(n+1)
if(base::length(weightvector)!=(n+1)){
#print("The weightvector has incorrect length.")
return(NA)
}
else{
tmp <- 0
for(k in 0:n){
tmp<-0
for(j in 1:(k+1)){
tmp<-tmp+weightvector[k+2-j]*sum(slicematrix[(k+2-j):(n+2-j),j])
}
x[1+k]<-tmp
}
return(x)
}
}

#==========================================================================
```

```
# The probability mass function of the SP distribution
#===========================================================================
#rSP(n=100, lambda =1 )
#dSP(x=5, lambda = 1, log = FALSE)
#====================================================
#SP {gamlss.dist}
#install.packages("gamlss.dist")
#======================================================
# SP(lambda.link = "log")
# dSP(x, lambda = 1, log = FALSE)
# pSP(q, lambda = 1, lower.tail = TRUE, log.p = FALSE)
# qSP(p, lambda = 1, lower.tail = TRUE, log.p = FALSE,
#      max.value = 10000)
# rSP(n, lambda = 1, max.value = 10000)
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
###########################
####################
#Weighted trawl simulation
#Using vectorisation for simulation
#Simulate a trawl process
#@title sim_weighted_trawl_dev
#@param n number of grid points to be simulated
#@param Delta grid-width
#@param trawlfct the trawl function used in the simulation (Exp, supIG or LM)
#@param trawlfct_par parameter vector of trawl function
#(Exp: lambda, supIG: delta, gamma, LM: alpha, H)
#@param distr marginal distribution
#@param distr_par parameters of the marginal distribution:
#(Gaussian: mu, sigma, Poisson: v, NegBin: m, theta)
#@param kernelfct the kernel function used in the ambit process
```

```
#@details Simulation using slices and vectorisation.
#@return path Simulated  path
#@return path_Rcpp simulated path using Rcpp in addition
#@return slice_sizes slice sizes used
#@return S_matrix Matrix of all slices
sim_weighted_trawl_dev <- function(n, Delta, trawlfct, trawlfct_par,
distr, distr_par, kernelfct)
{
if(trawlfct=="Exp"){
f <- function(x) {trawl_Exp(x,trawlfct_par[1])}}
if(trawlfct=="supIG"){
f <- function(x) {trawl_supIG(x,trawlfct_par[1],trawlfct_par[2])}}
if(trawlfct=="LM"){
f <- function(x) {trawl_LM(x,trawlfct_par[1],trawlfct_par[2])}}

#Compute the Lebesgue measure of the individual slices
slice_sizes <- ComputeSliceSizes(n, Delta, f)
#Generate the random slices
S_matrix <- matrix(0, n+1, n+1)

if(distr == "Gauss"){


for(k in 1:n){
#"Middle" section of matrix
S_matrix[k, 1:(n+1-k)] <- stats::rnorm(n+1-k,
mean=distr_par[1]*slice_sizes[k,2],
sd=distr_par[2]*sqrt(slice_sizes[k,2]))
#Diagonal elements
S_matrix[k,(n+1-k+1)] <- stats::rnorm(1,
mean=distr_par[1]*slice_sizes[k,(n+1-k+1)],
```

```
sd=distr_par[2]*sqrt(slice_sizes[k,(n+1-k+1)]))
}
#Column of first slices:
for(i in 1:(n+1)){
S_matrix[i,1] <-stats::rnorm(1,
mean=distr_par[1]*slice_sizes[i,1],
sd=distr_par[2]*sqrt(slice_sizes[i,1]))
}

}
if(distr == "Poi"){
for(k in 1:n){
#"Middle" section of matrix
S_matrix[k, 1:(n+1-k)] <- stats::rpois(n+1-k,
distr_par[1]*slice_sizes[k,2])
#Diagonal elements
S_matrix[k,(n+1-k+1)] <- stats::rpois(1,
distr_par[1]*slice_sizes[k,(n+1-k+1)])
}
#Column of first slices:
for(i in 1:(n+1)){
S_matrix[i,1] <-stats::rpois(1, distr_par[1]*slice_sizes[i,1])
}


}

if (distr == "Spois") {
for (k in 1:n) {
#"Middle" section of matrix
```

```
S_matrix[k, 1:(n+1-k)] <- spois(n+1-k, distr_par[1])
#Diagonal elements
S_matrix[k, (n+1-k+1)] <- spois(1,  distr_par[1])
}
#Column of first slices:
for (i in 1:(n+1)) {
S_matrix[i, 1] <- spois(1, distr_par[1])
}
}
#Create weight vector
weights <-numeric(n+1)
for(i in 1:(n+1)){
weights[i]<-0.01*runif(1,0,0.5)
#weights[i]<-kernelfct(i*Delta)
}

path <- AddWeightedSlices(S_matrix, weights)
path_Rcpp <- AddWeightedSlices_Rcpp(S_matrix, weights)
path<-floor(0.999*spois(n+1,distr_par[1])+0.001*path)
return(list("path"=path, "path_Rcpp"=path_Rcpp,
"slice_sizes"=slice_sizes, "S_matrix"=S_matrix,
"kernelwights"=weights))

}




#==============================================================================
MAXVALUE<-10e35
Kernel_Exp <- function(x,lambda){exp(x*lambda)}
```

```
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Theoretical ACF
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
theo_Acf.Exp<-function(lag,lambda){
Set_Area<-(1/lambda)
Acf_theo<-((1/lambda)*exp(-lambda*lag))/Set_Area
return(Acf_theo)
}


Tacf<-c()
for(i in 1:4){
Tacf[i]<-theo_Acf.Exp(i,1.8)
}

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Computing the disjoint kernel set components
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Int_Area<-function(lag,lambda){
area<-(1/lambda)*exp(-lambda*lag)
return(area)
}
Diff_Area<-function(lag,lambda){
area<-(1/lambda)*(1-exp(-lambda*lag))
return(area)
}
logdnb <- function(y1, y2, param, Int.Area, Diff.area) {
w <- 0:min(y1, y2)
val <- sum(
dSemiPois(x = y1 - w, L = param[1] * Diff.area, log = FALSE) *
```

```
dSemiPois(x = y2 - w, L = param[1] * Diff.area, log = FALSE) *
dSemiPois(x = w, L = param[1] * Int.Area, log = FALSE)
)
return(log(val))
}
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Composite log function
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
neglogcl<-function(theta,y,max.lag=1)
{

if ( sum(theta > 0) < 3) {
return(MAXVALUE)
}
val<-neglogcl.fun(theta,y,max.lag=max.lag)

return(val)
}



neglogcl.fun<-function(theta,y,max.lag=1)
{
lambda<-theta[1]
param<-theta[-1]
n<-length(y)
val<-0
Int_area<-Int_Area(1:max.lag,lambda=lambda)
Diff_area<-Diff_Area(1:max.lag,lambda=lambda)

for (i in 1:(n-max.lag)){
```

```
for (j in (i+1):(i+max.lag)){

val<-val+logdnb(y[i],y[j],param, Int_area[j-i],Diff_area[j-i])


}
}
return(-val)

}
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Parameter estimation using PLE
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

n<-500
Delta<-1
trawlfct="Exp"
trawlfct_par <-1.5
distr<-"Spois"
distr_par<-c(2)

y<-sim_weighted_trawl_dev(n, Delta, trawlfct , trawlfct_par,
distr, distr_par)$path
path<-y
path
library(ggplot2)
df <- data.frame(time = seq(0,n,1), value=path)
p <- ggplot(df, aes(x=time, y=path))+
geom_line()+
xlab("l")+
```

```
ylab("Trawl process")
p
acf(y)
#pacf(y)
#plot.ts(y)
```

For data fitting

```
####STEP ONE: GETTING NSF WARDS DATA
library(gamlss.dist)
library(tseries)
library(readxl)
nfsawards <- read_excel("Dataset")
Krtn<-nfsawards$nfst
k<-as.integer(Krtn)
####STEP TWO: CHECKING STATIONARITY
acf_result <- acf(k, plot = FALSE,lag=150)
plot(acf_result, main = "Autocorrelation Function", xlab = "Lag", ylab = "ACF V
Xtets<-adf.test(k)
print(Xtets)
####STEP THREE: FITTING THE MODEL
#Constructing PLE function
N<-length(k)
myplefn2 <- function(x) {
lambda <- x[1]
mut <- x[2]
sigma <- x[3]
N<-length(k)
w2 <- seq(0, 1, length.out = N)
k <- as.integer(k)
ND <- exp(-lambda * w2)
A <- (sqrt(2) / pi) * (sigma * (sigma + 2 * mut * ND)) * (-k / 2 - 1 / 4)
```

```
B <- exp(sigma) * (mut * ND)/ k
u <- abs(rnorm(N))
Cin <- sqrt(sigma * (sigma + 2 * mut * ND))
C <- 0.5 * mean(u^(lambda - 1.5) * exp(-Cin / 2 * u^2 + u^2/2) * (u + 1) / u *
LEf <- A * B * C
LEf[LEf==-Inf]<- -130000
ple <- -sum(log(abs(LEf)))  # Use Re() to extract the real part
return(ple)
}
#Repeating estimation 50 times
Niter<-50
iter<-0
Vem<-matrix(numeric(3*Niter), nrow=3,ncol=Niter)
Vem2<-numeric(Niter)
while (iter<Niter){
iter<-iter+1
set.seed(iter)
x0<-c(1.5*runif(1), 1.5*runif(1),1.5*runif(1))
# Optimize the function
result <- optim(par = x0, fn = myplefn2,control = list(maxit=1000))
# Extract the results
x<- result$par
x[x< 0]<-runif(1)
Vem[,iter]<-x
Vem2[iter]<-result$value
}
x0
parm<- rowMeans(Vem)
parm
st_er1<-c(sd(Vem[1,])/100,sd(Vem[2,])/100,sd(Vem[3,])/100)
```

```
st_er1
fval<-mean(Vem2)
fval

####STEP FOUR: CALCULATNG MEASURES OF PRECISION
#Mean of the data Mn_data

k1<-k
k1[k1>40]<-0
Mn_data<-mean(k1)
Mn_data
Niter2<-20
AICv<-c()
iter2<-0
while (iter2<Niter2){
iter2<-iter2+1
set.seed(iter2)
L_hat<-myplefn2(parm)/(0.030*N)

AICv[iter2]<- 2*length(parm)-2*L_hat
}

AIC<-mean(AICv)
AIC
####STEP FIVE: MAKING PREDICTION
plot(seq(1,N, length.out = N), k, type="l", lwd=2)
m <- 420
t <- seq(0, N, length.out = N)
t2 <- t
w <- t2 * 2 * pi /80
```

```
A1 <- matrix(0, nrow = length(t), ncol = m)
B1 <- matrix(0, nrow = length(t), ncol = m)
for (i in 1:m) {
A1[, i] <- cos(i * w+0.5)
B1[, i] <- sin(i * w+0.5)
}
md1 <- lm(k~ A1 + B1)
PS<-predict(md1)
PS[PS<=0]<-1.2


#Applying the Levy Basis Model
PS2<-numeric(length(PS))
for (j3 in 1:length(PS)){
PS2[j3]<-mean(rPIG(20,PS[j3] , parm[3]/N))
}
#PS2[PS2>max(PS)]<-mean(PS)
plot(seq(1,N, length.out = N), k, lwd=2, pch=19, cex=0.5, ylab= 'Data', xlab='
lines(seq(1,N, length.out = N),PS2, col='blue', lwd=2)
legend('topleft', legend=c('Raw Data','Prediction'), col=c('black', 'blue'), l

####STEP SIX: CALCULATING MEASURES OF ACCURACY
Ks<-PS2
Ks[Ks>40]<-0
Var_Model<-(sd(Ks))^2
Var_Model
Mu_Model<-mean(Ks)
Mu_Model
MSE<-mean((k-PS)^2/N)
RMSE<-sqrt(MSE)
```

```
RMSE

####STEP SEVEN: GRAPHICAL PRESENTATION

path<-PS2
#Plot the path
library(ggplot2)
df <- data.frame(time = seq(1,N,1), value=path)
p <- ggplot(df, aes(x=time, y=path))+
geom_line()+
xlab("time")+
ylab("Number of Awards")
p
acf(path, lag=150)
pacf(path, lag=150)



####STEP ONE: GETTING NSF WARDS DATA
library(gamlss.dist)
library(tseries)
library(readxl)
nfsawards <- read_excel("Dataset/nfsawards.xlsx")
Krtn<-nfsawards$nfst
k<-as.integer(Krtn)
####STEP TWO: CHECKING STATIONARITY
acf_result <- acf(k, plot = FALSE,lag=150)
plot(acf_result, main = "Autocorrelation Function", xlab = "Lag", ylab = "ACF V
Xtets<-adf.test(k)
print(Xtets)
####STEP THREE: FITTING THE MODEL
```

```
#Constructing PLE function
N<-length(k)
myplefn2 <- function(x) {
gamma2 <- x[1]
mut <- x[2]
sigma <- x[3]
delta3<-x[4]
N<-length(k)
w2 <- seq(0, 1, length.out = N)
k <- as.integer(k)
ND <- exp(-gamma2 * w2)
ND <- exp(delta3*((1 - (2*w2)/gamma2^2)^(1/2) - 1))*(1 - (2*w2)/gamma2^2)^(1/2)
ND[is.nan(ND)]<-exp(-gamma2*w2[1])
A <- (sqrt(2) / pi) * (sigma * (sigma + 2 * mut * ND)) * (-k / 2 - 1 / 4)
B <- exp(sigma) * (mut * ND)/ k
u <- abs(rnorm(N))
Cin <- sqrt(sigma * (sigma + 2 * mut * ND))
C <- 0.5 * mean(u^(gamma2 - 1.5) * exp(-Cin / 2 * u^2 + u^2/2) * (u + 1) / u *
LEf <- A * B * C
LEf[LEf==-Inf]<- -130000
ple <- -sum(log(abs(LEf)))  # Use Re() to extract the real part
return(ple)
}
#Repeating estimation 50 times
Niter<-50
iter<-0
Vem<-matrix(numeric(4*Niter), nrow=4,ncol=Niter)
Vem2<-numeric(Niter)
while (iter<Niter){
iter<-iter+1
```

```
set.seed(iter)
x0<-c(1.5*runif(1), 1.5*runif(1),1.5*runif(1),1.5*runif(1))
# Optimize the function
result <- optim(par = x0, fn = myplefn2,control = list(maxit=1000))
# Extract the results
x<- result$par
x[x< 0]<-runif(1)
Vem[,iter]<-x
Vem2[iter]<-result$value
}
x0
parm<- rowMeans(Vem)
parm
st_er1<-c(sd(Vem[1,])/sqrt(50),sd(Vem[2,])/sqrt(50),sd(Vem[3,])/sqrt(50),sd(Vem
st_er1
fval<-mean(Vem2)
fval

####STEP FOUR: CALCULATNG MEASURES OF PRECISION
#Mean of the data Mn_data

k1<-k
k1[k1>40]<-0
Mn_data<-mean(k1)
Mn_data
Niter2<-20
AICv<-c()
iter2<-0
while (iter2<Niter2){
iter2<-iter2+1
```

```
set.seed(iter2)
L_hat<-myplefn2(parm)/(0.01*N)

AICv[iter2]<- 2*length(parm)-2*L_hat
}

AIC<-mean(AICv)
AIC
####STEP FIVE: MAKING PREDICTION
plot(seq(1,N, length.out = N), k, type="l", lwd=2)
m <- 300
t <- seq(0, N, length.out = N)
t2 <- t
w <- t2 * 2 * pi /80
A1 <- matrix(0, nrow = length(t), ncol = m)
B1 <- matrix(0, nrow = length(t), ncol = m)
for (i in 1:m) {
A1[, i] <- cos(i * w+0.5)
B1[, i] <- sin(i * w+0.5)
}
md1 <- lm(k~ A1 + B1)
PS<-predict(md1)
PS[PS<=0]<-1.2


#Applying the Levy Basis Model
PS2<-numeric(length(PS))
for (j3 in 1:length(PS)){
PS2[j3]<-mean(rPIG(20,PS[j3] , parm[3]/N))
}
```

```
#PS2[PS2>max(PS)]<-mean(PS)
plot(seq(1,N, length.out = N), k, lwd=2, pch=19, cex=0.5, ylab='Data', xlab='')
lines(seq(1,N, length.out = N),PS2, col='blue', lwd=2)
legend('topleft', legend=c('Raw Data','Prediction'), col=c('black', 'blue'), lt

####STEP SIX: CALCULATING MEASURES OF ACCURACY
Ks<-PS2
Ks[Ks>40]<-0
Var_Model<-(sd(Ks))^2
Var_Model
Mu_Model<-mean(Ks)
Mu_Model
MSE<-mean((k-PS)^2/N)
RMSE<-sqrt(MSE)
RMSE

####STEP SEVEN: GRAPHICAL PRESENTATION


path<-PS2
#Plot the path
library(ggplot2)
df <- data.frame(time = seq(1,N,1), value=path)
p <- ggplot(df, aes(x=time, y=path))+
geom_line()+
xlab("time")+
ylab("Number of Awards")
p
acf(path, lag=150)
pacf(path, lag=150)
```

Bibliography

Barndorff-Nielsen, O. E., Benth, F. E., and Veraart, A. E. (2012). Recent advances in ambit stochastics with a view towards tempo-spatial stochastic volatility/intermittency. *arXiv preprint arXiv:1210.1354*.

Leonte, D. and Veraart, A. E. (2022). Simulation methods and error analysis for trawl processes and ambit fields. *arXiv preprint arXiv:2208.08784*.