Appendix A: Supplemental Information

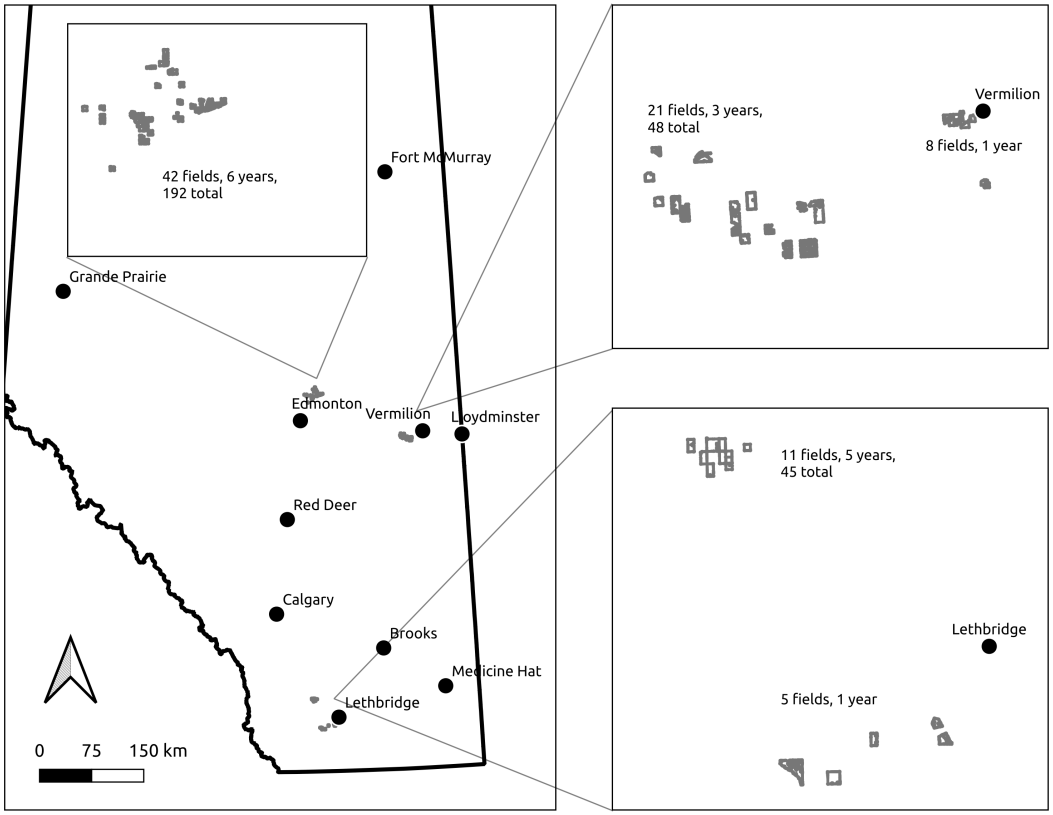

Figure S1: Locations of fields that data were collected from.

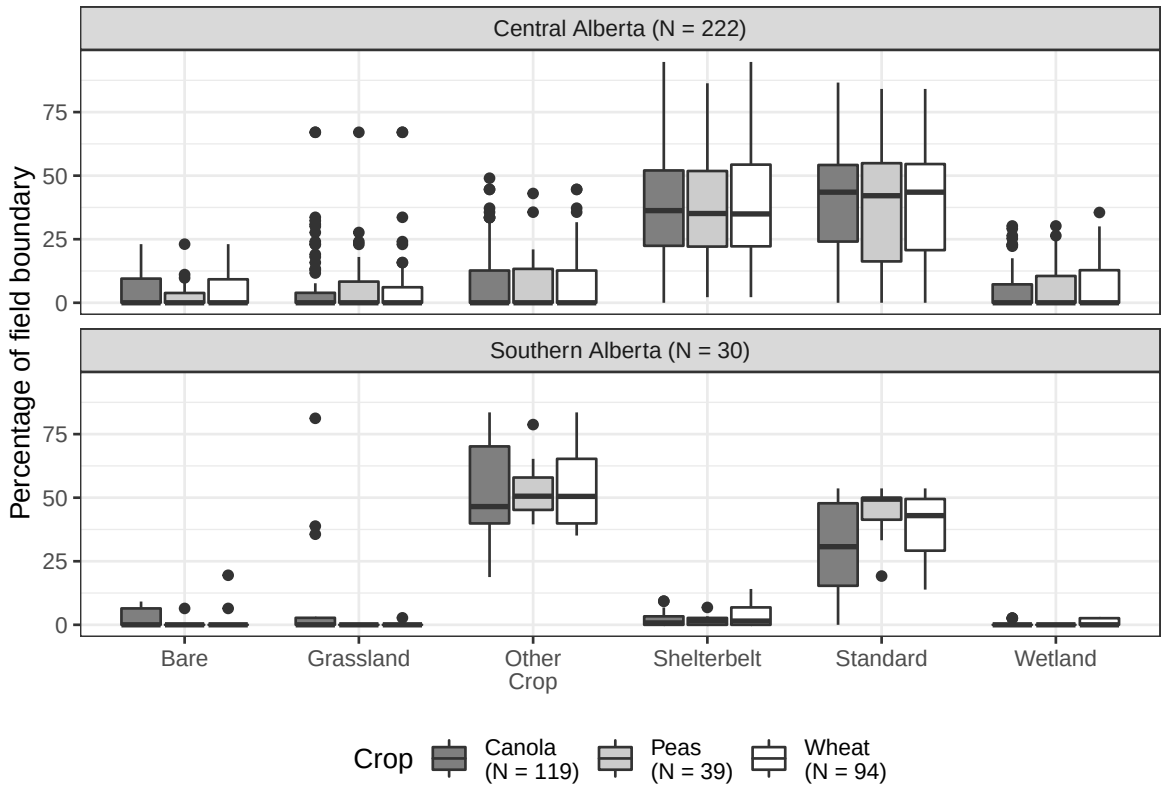

Figure S2: Boxplot of field boundary compositions across sampled fields. The x-axis indicates the boundary type, while the y-axis indicates the percentage of all field boundaries covered by that boundary type.

Appendix B: R Code for Data Filtering

```
# Helper functions -----------------

#"Inlier" spatial filtering procedure from Vega et al 2019, rewritten to work
#with sf + dplyr. Returns boolean
vegaFilter <- function(data,ycol,pvalCutoff=0.05,nDist=40){
  require(sf)
  require(sp)
  require(spdep)
  require(tidyverse)

  if(!any(class(data) %in% 'sf')) stop('Dataframe must be sf object')

  coords <- st_coordinates(data) #Get coordinates in matrix form

  #Get neighbourhood weights from 0 to ndist meters
  nWeights <- dnearneigh(coords,0,nDist)

  if(any(sapply(nWeights,length)==1)){
    warning('Some points had no neighbours and were removed')
  }

  #Get neighbourhood indices for each point (which other points are in this
  #point's neighbourhood?)
  nIndices <- nb2listw(nWeights, style = "W",zero.policy = TRUE)

  yield <- pull(data,{{ycol}}) #Get yield data column

  #Local Moran's I for each neighbourhood
  LM <- localmoran(yield,nIndices,p.adjust.method="bonferroni",
                   alternative ="less")

  results <- data.frame(LM) %>%
    rename('pval'=contains('Pr.z.')) %>%
    #Filter negative Ii and pvals < 0.05
    mutate(keepThese= Ii > 0 | pval > pvalCutoff) %>%
    #NAs (points had no neighbours)
    mutate(keepThese=ifelse(is.na(keepThese),FALSE,keepThese))

  ret <- pull(results)
  return(ret)
}

#Function to filter anything above a certain z-score. Returns boolean
ZscoreFilter <- function(x,z=3){
  xmean <- mean(x,na.rm=TRUE)
  xsd <- sd(x,na.rm=TRUE)
  zscore <- abs(((x-xmean)/xsd))
  zscore<z
}
```

```r
#Function to filter anything above certain quantiles. Returns boolean
QuantileFilter <- function(x,quant=0.99){
  l <- c((1-quant)/2,1-(1-quant)/2) #Symmetric quantiles
  x>quantile(x,l[1]) & x<quantile(x,l[2])
}

#Function to filter large changes in bearing. Returns boolean unless
#returnDiffs==TRUE
bearingFilter <- function(bearing,q=NULL,z=NULL,returnDiffs=FALSE){
  if(!xor(is.null(q),is.null(z))&!returnDiffs) stop('Input quantiles or Z-score')
  #Difference in compass bearings (in degrees)
  bearingDiff <- function(x1,x2){
    x <- x1-x2
    x <- ifelse(abs(x)>180,x-(360*sign(x)),x) #Angle differences can't be >180
    return(x)
  }

  #Looks 1 point ahead and behind
  bd <- cbind(bearingDiff(lag(bearing),bearing),
              bearingDiff(lead(bearing),bearing))
  #Maximum bearing difference ahead and behind
  bd <- apply(bd,1,function(x) max(abs(x),na.rm=TRUE)*sign(x[which.max(abs(x))]))

  if(returnDiffs) return(bd) #Return bearing differences only, without filtering

  if(!is.null(q)){
    ret <- QuantileFilter(bd,q=q)
  } else {
    ret <- ZscoreFilter(bd,z=z)
  }
  return(ret)
}


#Positional difference filter - filters out very distant and very close points.
#Returns boolean unless returnDiffs==TRUE
posFilter <- function(data,q=NULL,returnDiffs=FALSE){
  require(sf)
  if(is.null(q)&!returnDiffs) stop('Input upper quantile')

  if(units(st_distance(data[1,],data[2,]))$numerator!='m'){
    warning('Position differences not in meters')
  }
  coords <- st_coordinates(data) #Get coordinates
  #Distances between points
  pdiff <- sapply(1:(nrow(coords)-1),function(i) as.numeric(dist(coords[i:(i+1),])))
  pdiff <- cbind(c(pdiff,NA),c(NA,pdiff)) #Forward and backward lags
  pdiff <- apply(pdiff,1,max,na.rm=TRUE) #Maximum distance ahead and behind
  if(returnDiffs){
    return(pdiff)
  } else {
    return(QuantileFilter(pdiff,q)) #Note: uses 2-sided quantiles
```

```
  }
}

#Filter for (forward and backward) lagged speed differences. Returns boolean
dSpeedFilter <- function(speed,l=c(-1,1),perc=0.2){

  #Matrix of % diffs
  llist <- sapply(l,function(x) (lag2(speed,x)-speed)/lag2(speed,x))

  #Are any lagged speed values > percent change threshold?
  ret <- !apply(llist,1,function(y) any(abs(y)[!is.na(y)]>perc))

  return(ret)
}

#Overloaded lag function that takes negative values
lag2 <- function(x,n){
  require(dplyr)
  if(n==0) {
    return(x) #No lag
  } else if(n>0){
    lag(x,n) #Positive lag
  } else {
    lead(x,abs(n)) #Negative lag
  }
}

#Filtering pipeline  -------------------------

library(tidyverse)
library(sf)

#Note: dat == sf dataframe of yield data with columns: DryYield (T/ha),
#TrackAngle (compass bearing), Speed (km/hr), cropType (Wheat, Canola, Peas)

dat <- dat %>%
  #Vega et al 2019 spatial "inlier" filter - takes a few minutes...
  mutate(vegaFilt = vegaFilter(.,DryYield,nDist = 30))

dat <- dat %>%
  #"BS" filter - removes values >10.75 T/ha for wheat, >8 T/ha for peas & canola
  mutate(noBS = DryYield<ifelse(cropType=='Wheat',10.75,8)) %>%
  #Trim dry yield outliers
  mutate(Qfilt = QuantileFilter(DryYield,q=0.98)) %>%
  #Trim extreme bearing changes (turning)
  mutate(bFilt = bearingFilter(TrackAngle,q=0.98)) %>%
  #Trim absolute speed outliers
  mutate(speedFilt = QuantileFilter(Speed,q=0.98)) %>%
  #Trim speed differences (>20% change 2 steps forward and backward, suggested
  #by Lyle et al 2014)
  mutate(dSpeedFilt = dSpeedFilter(Speed,l=c(-2,-1,1,2),perc = 0.2)) %>%
  #Trim points that are far away from eachother
```

```
mutate(posFilt = posFilter(.,q=0.98)) %>%
#Combine filter criteria
mutate(allFilt = noBS & vegaFilt & Qfilt & bFilt & speedFilt &
         dSpeedFilt & posFilt) %>%
#Remove filtered values
filter(allFilt)
```

Appendix C: R Code for Modelling

```
# Fit models ------------------------------------------------------------------

library(mgcv)

#Note: dat == dataframe of yield data with columns: DryYield (T/ha), dist
#(distance to nearest field boundary, m), boundaryType (type of boundary,
#factor), E,N (easting and northing distance from centre of field, m)

#Uses thin-plate splines with extra shrinkage (bs = 'ts'), with 5 basis
#functions for distance smoothers and 80 basis functions for spatial smoothers
#(see Wood 2017 for more details)

#Variance smoothers
f2 <- '~ s(dist,k=5,bs="ts",by=boundaryType) + s(E,N,k=80)'
#Mean smoothers
f <- 'sqrt(DryYield)~ s(dist,k=5,bs="ts",by=boundaryType) + s(E,N,k=80)')
#Combine into a list
flist <- list(as.formula(f),as.formula(f2))

#Fit model - takes about 45 mins for a 50000 point dataset using an AMD Ryzen 7
#3800X 8-core processor
mod <- gam(flist,data=dat,family=gaulss())
```