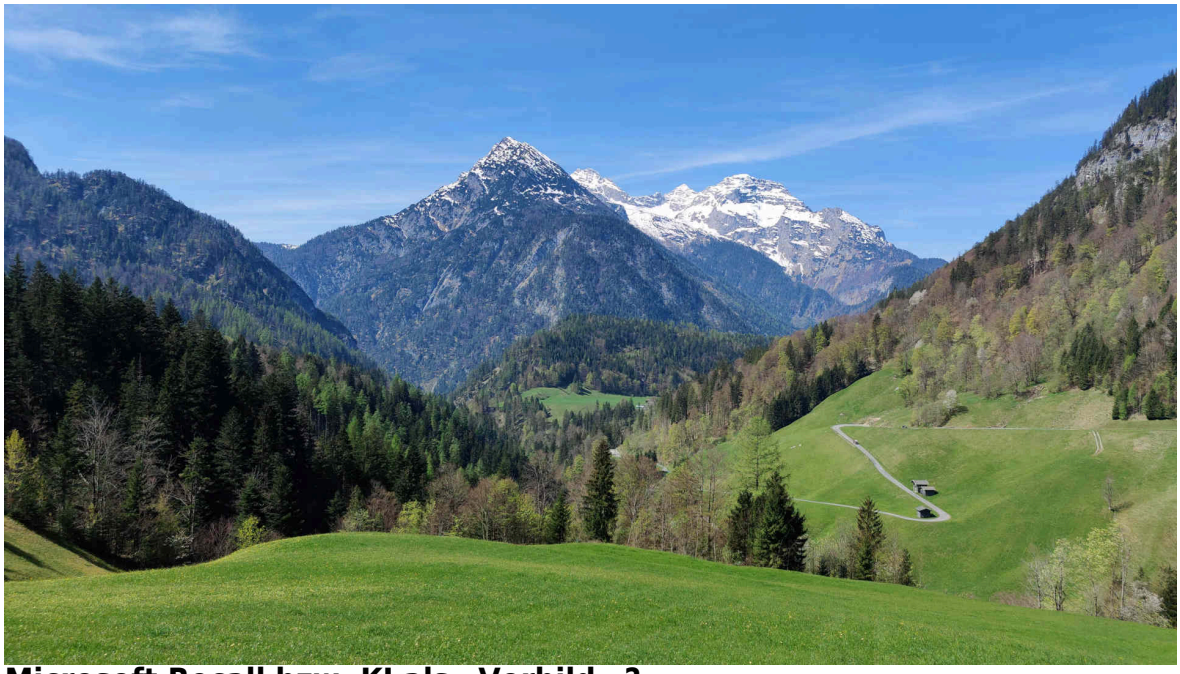
Bildschirmfotos aufzeichnen und PDF-Dateien erstellen

Egg, 12. Juni 2024: *Allumfassend und überall scheint es aktuell um künstliche Intelligenz (KI) zu gehen. Einige Bekanntheit erlangt aktuell Microsoft mit dem neu für Windows 11 angekündigten Dienst Recall, der sozusagen das Gedächtnis der Nutzer/innen auf ihrem Computer werden soll. Dabei werden alle x-Sekunden Bildschirmfotos erstellt. Diese Daten werden im Hintergrund KI «gerecht» aufbereitet, sodass spätere Recherchen möglich sind. Mit Capt2PDF gibt es neu ein Tool für Linux. Im Unterschied zu Microsoft Recall arbeitet Capt2PDF auf dem lokalen Desktop, die Kontrolle der Daten verbleibt bei den Nutzer/innen.*

Darum gibt es Capt2PDF neu für den Desktop

Um Bildschirminhalte aufzuzeichnen, gibt es zwei grundsätzliche Ansätze. Meistens wird aktuell OBS (Open Broadcasting System) verwendet. Dabei werden Filmdateien (inkl. Ton) der Bildschirmaktivität erstellt. Mittels Text- und Spracherkennung können daraus durchsuchbare PDF-Dateien erstellt werden. Mit OBS erstelle Filmdateien benötigen viel Platz, werden doch pro Sekunde ca. 30 Bilder aufgezeichnet. Der Platzbedarf pro Minute bewegt sich im Bereich von mehreren Dutzend MByte. Die spätere Text- und Spracherkennung erfordern viel Leistung, ein Tool, dass diesen Job automatisch erledigt, gibt es aktuell nicht.

Als Alternative bietet sich der klassische Ansatz an. Wohl auf jedem Linux-Desktop können mit ‚PrintScreen' (Taste ganz oben, dritte von rechts) Bildschirmkopien erstellt werden. Geht es darum, einige nachfolgende Abläufe (z.B. im Support) zu dokumentieren, wird das Unterfangen bald arbeitsintensiv. Die einzelnen Bilddateien müssen zu einer Datei umgewandelt werden. Meistens wird dabei eine PDF-Datei erstellt. Damit der Inhalt durchsuchbar ist, muss eine Texterkennung gestartet werden etc. etc.



Microsoft Recall bzw. KI als «Vorbild» ?
An sich nimmt Microsoft Recall genau dieses Anliegen auf. Das automatische Aufzeichnen von Inhalten ist aber nicht in Ansätzen neu. Neu ist einfach, dass das Erstellen der Bildschirmkopien quasi in Echtzeit automatisiert so gemacht wird, dass sich die Daten durchsuchen lassen.
Dass dazu in irgendeiner Art und Weise die Cloud zum Einsatz kommen soll/muss, bleibt ein Rätsel. Ganz offensichtlich geht es wohl eher darum, dass die Hersteller möglichst viele Daten «abkupfern» wollen.
Leider ist dies längst nicht nur bei Microsoft bzw. Windows der Fall. Apple hat gerade angekündigt, dass die Produkte allumfassend mit KI-Funktionen ausgestattet werden. Gemäss «Artikel» zur **Apple-Präsentation vom 10.6.24 erschien beim Schweizer Fernsehen (SRF) ein Beitrag,** siehe auch das mit **Capt2PDF-erstellte Dokument hier.** Daraus Zitat:
Die Funktionen seien tief in die Betriebssysteme für iPhone, Mac und iPad eingebettet worden, betonte Software-Chef Craig Federighi. Damit hätten Apples KI-Modelle Zugang zu nötigen Informationen der Nutzerinnen und Nutzer, um nützlich für sie zu sein. Viele der Modelle liefen direkt auf den Geräten, betonte Federighi. Bei Bedarf werde auch die Cloud zugeschaltet – aber mit einer verschlüsselten Verbindung. Die Apple-Software entscheide von Fall zu Fall, ob eine Aufgabe lokal oder über die Cloud ausgeführt werden sollte.
Für all jene, die hier nicht **«Rot»** sehen, sei folgendes angemerkt.
1. Ein Betriebssystem hat nicht «nur» Zugang zu nötigen Informationen, es hat immer Zugriff zu allen Daten. Wie soll sonst z.B. überhaupt eine Datei gespeichert werden können? Dies bedeutet aber keinesfalls, dass eine KI arbeiten muss.
2. Ist ein Betriebssystem nicht nutzbar, ergibt es keinen Sinn. Um arbeiten zu können, ist allerdings keine KI notwendig, weder lokal noch in der Cloud.
3. Keine Software entscheidet von Fall zu Fall, ob persönliche Daten in die Cloud ausgelagert werden. Bei jeder Software ist von Anfang an klar, was wo und wie abläuft. Für wie blöd werden Nutzer/innen gehalten? Wie «beschränkt» sind Medien, wenn ein solcher Irrsinn so publiziert wird?
Abgesehen davon, dass es mit dem obigen Beispiel darum geht, aufzuzeigen, dass unter dem Begriff KI sehr viel Unfug getrieben wird, kann mit diesem Artikel auch aufgezeigt werden, wie einfach Capt2PDF in der Anwendung ist.
Der Quellcode von Capt2PDF findet sich unten. Das Programm kann auf einem Terminal mit **perl capt2pdf 5** (Aufzeichnung mit Intervall von 5 Sekunden)

gestartet werden. Mit **perl capt2pdf 0** wird die Aufzeichnung gestoppt. Diese beiden Befehle werden sinnvollerweise auf eine Tastenkombination gelegt. Auf dem AVMultimedia-Desktop steht dafür die Tastenkombination **Ctrl+PrintScreen zum Starten der Aufzeichnung** bzw. **Shift+Ctrl+PrintScreen zum Beenden der Aufzeichnung** zur Verfügung. Dazwischen wird der Bildschirm in Abständen von 5 Sekunden aufgezeichnet. Nach Beenden des Programms wird nach einigen wenigen Sekunden die fertig durchsuchbare PDF-Datei auf dem Desktop dargestellt. Einfach, oder?

Braucht Capt2PDF eine KI oder eine Cloud ?
Nicht in Ansätzen. Capt2PDF **(Zip-Datei mit Sourcen hier)** arbeitet 100% lokal, eine jede halbwegs moderne CPU ist in der Lage, diesen Job zu erledigen. Capt2PDF umfasst aktuell ca. 260 Zeilen, die hier publiziert werden. Wer nicht ganz so technisch unterwegs ist bzw. sich dafür nicht interessiert, möge einfach beim nächsten Absatz weiterlesen.

```
#!/usr/bin/perl
# capt2pdf (c) v0.2 - 2024-06-12 by Archivista GmbH, Urs
Pfister, GPLv2
# program to capture every x seconds a screenshot, check for
at least 1%
# difference in the screens (compare from ImageMagick) and
if so, set
# resolution (convert from ImageMagick) and send it to
tesseract for
# background ocr (incl. balance checker for 1/2 of cpus),
needed programs:
# scrot|spectacle|gnome-
screenshot,compare,convert,pdftk,tesseract,zenity

use strict;
use constant PFAD0 => "/home/archivista/data";
use constant PFAD1 => PFAD0."/screens";
use constant PFAD => `echo -n \$HOME`.'/screens'; # home
folder / logs etc
if (-e PFAD0) { # arrange paths if it is
AVMultimedia/ArchivistaBox
  mkdir PFAD1 if !-e PFAD1;
```

```
  doit("ln -s ".PFAD1." ".PFAD) if !-e PFAD
} else { # all other we take screens folder in home
directory
  mkdir PFAD if !-e PFAD;
}
use constant RUNS => PFAD."/capt2pdf.wrk"; # running (has
screen pixels)
use constant OCRLANGS => PFAD."/capt2pdf.ocr"; # languages
for tesseract
use constant X11CAPT => "/usr/bin/scrot"; # X11 screen
caputre program
use constant X11OPT => ""; # options (last one needs to be
file flag)
use constant WAYCAPT => "/usr/bin/spectacle"; # WAYLAND
screen capture program
use constant WAYOPT => "-f -b -n -o"; # options, last one -o
for file name
use constant WAYCAPT2 => "/usr/bin/gnome-screenshot"; #
WAYLAND gnome
use constant WAYOPT2 => "-f"; # options, last one -o for
file name
use constant CONVERT => "/usr/bin/convert"; # set dpi
(resolution)
use constant COMPARE => "/usr/bin/compare"; # comparing
images (ImageMagick)
use constant PDFTK => "/usr/bin/pdftk"; # program to
compbine pdf pages together
use constant TESSERACT => "/usr/bin/tesseract"; # ocr
recognition
use constant TESSOCR => "eng+deu"; # default languages for
tesseract
use constant RESDPI => "300x300"; # set image res (if no at
all, set '')
use constant XRANDR => "/usr/bin/xrandr"; # get resolution
of screen (1st one)
use constant PDFVIEW => "/usr/bin/qpdfview"; # open file
with pdf viewer
use constant MSG => "/usr/bin/zenity"; # send message with
zenity
use constant NOTE => "/usr/bin/notify-send"; # send message
with notify-send
use constant TIT => "Capt2PDF"; # title for application
use constant START => "start..."; # start message
use constant STOP => "stop..."; # stop message
use constant PERL => "/usr/bin/perl"; # get resolution of
screen (1st one)
use constant D0 => " 2>/dev/null"; # suppress error messages
my $mode = shift; # 0=stopit, 2-60=startit, file=capture
(called from prg)
my $lang = shift; # language string for tesseract
my $job = $0;
die "$0 mode [lang] => 0=stop,2-60=capt.time,deu+eng=ocr
lang" if $mode eq "";
logit("$0 called with $mode");
if ($mode eq "0" || $mode eq "") {
  stopit(); # stop capturing and create pdf file
} elsif ($mode>=2 && $mode<=60) {
```

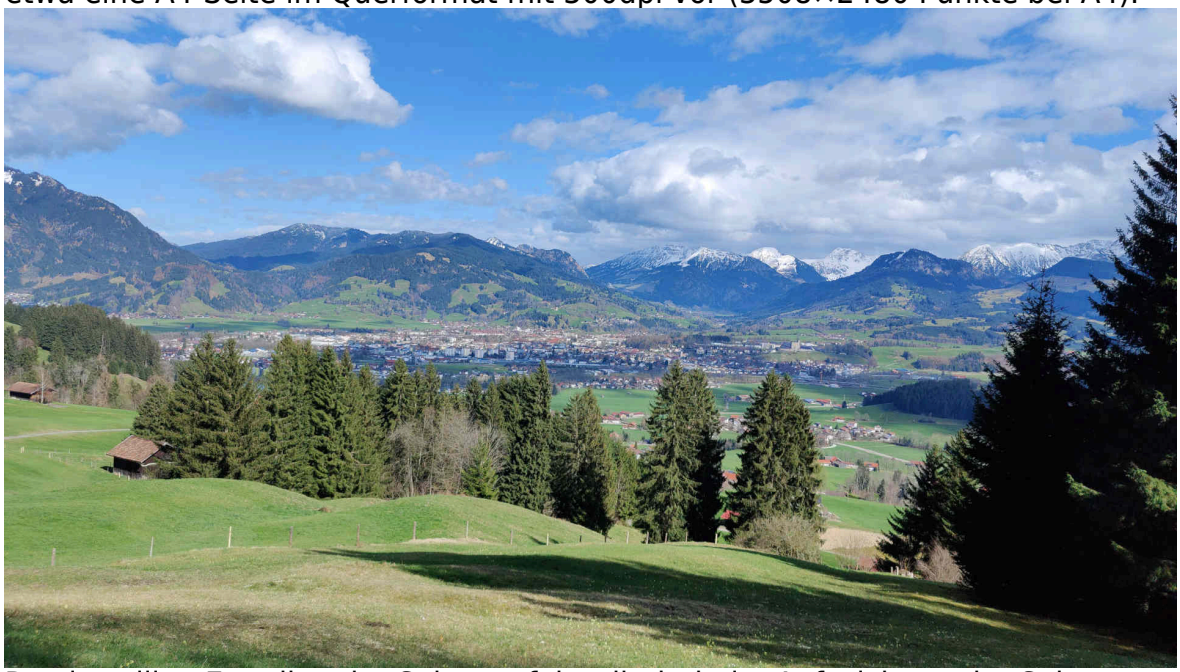
```
  startit($mode,$lang); # start capturing (must be between 2
and 60 seconds)
} elsif (-e "$mode") {
  capture($mode); # a new page hase arrived, so process it
}

sub doit { # process a system call and log results
  my ($cmd) = @_;
  my $res=system($cmd);
  logit("$res=>$cmd");
  return $res;
}

sub logit { # log file to get infos about what was done
  my ($msg) = @_;
  open(FOUT,">>".PFAD."/av.log");
  print FOUT "$msg\n";
  close(FOUT);
}

sub getpixels { # get pixel of your screen (if no xrandr =>
then fullhd)
  my $screenx = 1980; my $screeny = 1080;
  my $xrandr = XRANDR;
  if (-e $xrandr) {
    my $xrandr = `$xrandr | grep '*'`;
    $xrandr =~ /([0-9]+)(x)([0-9]+)/;
    if ($1>0 && $2 eq "x" && $3>0) {
      $screenx = $1;
      $screeny = $3;
    }
  }
  my $maxpixels = $screenx*$screeny;
  logit("resolution of screen:$screenx:$screeny");
  return $maxpixels;
}

sub writefile { # write simple text file
  my ($file,$cont) = @_;
  open(FOUT,">$file");
  print FOUT "$cont";
  close(FOUT);
}

sub stopit { # stop capturing and create pdf file
  my $pfad = PFAD; my $ocr = -1; my $capt = 0; my $first=1;
  doit("rm ".RUNS) if -e RUNS;
  noteit(STOP,5);
  for(my $c=2;$c<10;$c++) { # wait for termination of ocr
    $ocr = countJobs(); # default is tesseract
    my $conv = countJobs(CONVERT);
    my $comp = countJobs(COMPARE);
    if (iswayland() eq "") {
      $capt = countJobs(X11CAPT);
    } else {
      $capt = countJobs(WAYCAPT) if -e WAYCAPT;
      $capt = countJobs(WAYCAPT2) if -e WAYCAPT2;
```

```
    }
    logit("ocr jobs:$ocr--conv:$conv--comp:$comp--
capt:$capt");
    if ($ocr==0 && $conv==0 && $comp==0 && $capt==0) {
      last if $first==0; $first=0;
    }
    sleep $c;
  }
  if ($ocr>0) {
    logit("we stop waiting, $ocr ocrjobs left");
  } else {
    logit("ocr processing has ended");
  }
  my @files = <$pfad/av-*.pdf>;
  my $file = $files[0];
  if (-e $file) {
    $file =~ s/(av-)/screen-/g; # unique name so it is not
killed next time
    my $prg = PDFTK." ".join(" ",@files)." output
$pfad/all.pdf";
    doit($prg);
    if (-e "$pfad/all.pdf") {
      doit("rm -f $pfad/av-*");
      doit("mv $pfad/all.pdf $file");
    }
  }
  doit(PDFVIEW." $file".D0." &") if -e PDFVIEW && PDFVIEW ne
"" && -e $file;
  doit("rm ".OCRLANGS) if -e OCRLANGS;
}

sub startit { # start capturing of sceenshots
  my ($wait,$lang) = @_;
  my $pfad = PFAD;
  $lang = ocrlangs($lang);
  if (!-e RUNS && !-e OCRLANGS) {
    noteit(START,2);
    my $pixels = getpixels();
    writefile(RUNS,$pixels);
    writefile(OCRLANGS,$lang);
    doit("rm -f /tmp/av*.png");
    doit("rm -f /tmp/av*.gif");
    doit("rm -f $pfad/av-*");
  } else {
    logit("$0 already started");
  }
  my $file = "/tmp/av-current.png";
  $wait--; # 1 second pause between ending capturing and
post processing
  my $cmd = X11CAPT." ".X11OPT;
  if (iswayland() ne "") {
    $cmd = WAYCAPT." ".WAYOPT if -e WAYCAPT;
    $cmd = WAYCAPT2." ".WAYOPT2 if -e WAYCAPT2;
  }
  my $prg = "sleep 1;".PERL." $0 $file".D0;
  doit("while [ -e ".RUNS." ];do sleep $wait;$cmd $file
".D0.";$prg;done &");
```

```
}

sub ocrlangs { # check and combine ocr languages for
tesseract
  my ($lang) = @_;
  my $langs = "";
  my @langs = split(/\,/,$lang);
  foreach my $lng (@langs) {
    if ($lng eq "deu" || $lng eq "frk" || $lng eq "fra" ||
$lng eq "ita" ||
      $lng eq "spa" || $lng eq "nld" || $lng eq "eng" ||
$lng eq "deu-frak") {
      $langs .= "+" if $langs ne "";
      $langs .= $lng;
    }
  }
  $langs = TESSOCR if $langs eq "";
  return $langs;
}

sub capture { # process captured page (incl. check if we
have differences)
  my ($file2) = @_;
  my $file = "/tmp/av-".timestamp().".png";
  doit("rm -f $file") if -e $file;
  doit("mv $file2 $file") if -e $file2 && !-e $file;
  my $pfad = PFAD;
  my @lines = <"/tmp/av-*.png">;
  my $current = pop @lines;
  my $last = pop @lines;
  if ($last ne "" && $file eq $current) {
    logit("curr:$current");
    logit("last:$last");
    my $check = $last.".gif";
    my $cmp = COMPARE;
    my $pixeldiff= int `$cmp -metric AE -fuzz 5% $last $file
$check 2>&1`;
    my $pixeldiff1 = $pixeldiff*100; # scale 1% to 100% for
comapring
    my $chk = "cat ".RUNS.D0;
    my $maxpixels = `$chk`;
    if ($pixeldiff1>$maxpixels) {
      logit("SAVENEW with $pixeldiff");
      captureOCR($last);
    } else {
      logit("NEARTO with $pixeldiff");
    }
    doit("rm $last") if -e "$last";
    doit("rm $check") if -e "$check";
  }
}

sub captureOCR { # create a pdf page from current screen
shot
  my ($file) = @_;
  my $pfad = PFAD; my $langs = ""; my $chk = OCRLANGS;
  $langs = `cat $chk` if -e "$chk";
```

```
  $langs = TESSOCR if $langs eq "";
  my $cpus = `cat /proc/cpuinfo | grep processor | wc -l`;
  my $ocrjobs = countJobs();
  if ($ocrjobs*2<=$cpus) {
    my @parts = split(/\//,$file);
    my $fname = pop @parts;
    @parts = split(/\./,$fname);
    my $ext = pop @parts;
    my $fbase = join('.',@parts);
    if (!-e "$pfad/$fbase.pdf") {
      if (RESDPI ne "") {
        my $cmd = CONVERT." $file -density ".RESDPI." -units
pixelsperinch ".
          "$pfad/$fbase.jpg";
        doit($cmd);
        $file = "$pfad/$fbase.jpg" if -e "$pfad/$fbase.jpg";
      }
      my $cmd = TESSERACT." -l $langs $file $pfad/$fbase pdf
".D0." &";
      doit($cmd);
    }
  }
}

sub countJobs { # give back the number of tesseract sessions
  my ($job) = @_;
  $job = TESSERACT if $job eq "";
  my $jobs = `ps ax | grep $job | grep -v grep | wc -l`;
  chomp $jobs;
  $jobs=0 if $jobs eq "";
  return $jobs;
}

sub timestamp { # give back current time stamp
  my @t = localtime( time() );
  my $Y = $t[5] + 1900;
  my $M = sprintf( "%02d", $t[4]+1 );
  my $D = sprintf( "%02d", $t[3] );
  my $h = sprintf( "%02d", $t[2] );
  my $m = sprintf( "%02d", $t[1] );
  my $s = sprintf( "%02d", $t[0] );
  return $Y."-".$M."-".$D."_".$h."-".$m."-".$s;
}

sub iswayland { # check if it is wayland display manager
  return `echo -n \$XDG_SESSION_TYPE`;
}

sub noteit {
  my ($msg,$wait) = @_;
  if (-e NOTE) {
    $wait=$wait*1000;
    doit(NOTE." ".TIT." $msg -t $wait".D0);
  } elsif (-e MSG) {
    doit(MSG." --timeout=$wait --info --title=".TIT." --
text=$msg".D0);
  }
```

}
Das Programm wurde für AVMultimedia bzw. die ArchivistaBox entwickelt. Es sollte aber auch auf den meisten anderen Linux-Desktops laufen. Unter X11 kommt das Programm **scrot** für die Bildschirmkopien zur Anwendung, bei Wayland ist es **spectacle** oder **gnome-screenshot.**
Erwähnt an dieser Stelle sei, dass neben dem Capture-Tool minimal die Programme **compare, convert, tesseract und pdftk** vorhanden sein müssen.
Capt2PDF startet einen Einzeiler in der Bash, welcher solange Bildschirmkopien erstellt, bis Capt2PDF beendet werden soll. Nach dem Erstellen einer Bildschirmkopie wird jeweils Capt2PDF unter Angabe des aktuellen Abbilds gestartet und von dort aus im Hintergrund die Texterkennung (OCR) **tesseract.**
Beim ImageMagick-Hilfsprogramm **compare** geht es darum, herauszufinden, ob sich die aktuelle Bildschirmkopie von der letzten unterscheidet. Nur dann ergibt es Sinn, daraus eine durchsuchbare PDF-Datei zu erstellen. Aktuell müssen sich 1% der Pixel auf dem Bildschirm ändern, ansonsten wird die Kopie verworfen. Damit werden bei reinen Mausbewegungen oder bei aktualisierten Statusanzeigen keine neuen Screens bzw. PDF-Seiten erstellt.
Liegen mehr als 1% geänderte Pixel vor, wird **tesseract** aufgerufen. Bevor die Texterkennung gestartet wird, wird mit **convert** die gewünschte Auflösung (dpi) in der Bilddatei hinterlegt. 300dpi ergeben bei der Texterkennung (OCR) allseits gute Resultate. Bei einem 4K-Bildschirm mit 3860×2160 Pixel liegt plus/minus in etwa eine A4-Seite im Querformat mit 300dpi vor (3508×2480 Punkte bei A4).



Das jeweilige Erstellen der Seiten erfolgt direkt beim Aufzeichnen der Seiten. Damit bei langsamen CPUs (Prozessoren) keine «Überlastung» resultiert, prüft Capt2PDF die Anzahl der CPU-Kerne. Sofern mehr als die Hälfte der CPUs für **tesseract** im Einsatz stehen, erfolgt solange keine Texterkennung mehr, bis wieder genügend Ressourcen zur Verfügung stehen. Capt2PDF wird hier «etwas» vergesslich. Angemerkt sei aber, dass dies nur bei sehr alten Prozessoren auftritt, in aller Regel benötigt **tesseract** ca. 2 Sekunden für eine Seite.
Die entsprechenden PDF-Dateien werden im **Home-Verzeichnis** (z.B. /home/arcihvista) unter **screens** quasi als einzelne Seiten abgelegt. Wenn Capt2PDF beendet wird, kommt **pdftk** zum Zuge, um die PDF-Gesamtdatei zu erstellen. Dieser Vorgang dauert auch für Hunderte von Seiten kaum mehr als fünf bis zehn Sekunden. Zum Abschluss wird die durchsuchbare PDF-Datei mit dem Viewer **qpdfview** dargestellt.

Kleiner Werbespot an dieser Stelle: Wer eine weiteregehende Auswertung bzw. integrale Durchsuchbarkeit mehrerer PDF-Dateien benötogt, findet in der ArchivistaBox ein gutes Arbeitstier. Wer kein vollwertiges Dokumenten Management System (DMS) einsetzen mag, kann natürlich auch mehrere so erstellte durchsuchbare PDF-Dateien zu einer Datei zusammenfügen. Der entsprechende Aufruf ist einfach:

```
pdftk file1.pdf file2.pdf output allfiles.pdf
```

Capt2PDF ist selbstverständlich Open Source (GPLv2). Es wurde für die Linux-Distribution AVMultimedia (und damit auch für die ArchivistaBox) entwickelt und steht dort ab Version 2024/VI über die Tastenkombinationen Ctrl+PrintScreen bzw. Shift+Ctrl+PrintScreen zur Verfügung. Mit einer Grösse von aktuell ca. 260 Zeilen Code ist es ein handliches Tool, um z.B. bei einer Recherche im Internet Inhalte bequem und einfach erfassen zu könen. Anzumerken bleibt, dass so aufgezeichnete Inhalte nicht beliebig ins Internet gestellt werden dürfen (dazu unten gleich noch mehr).

Capt2PDF oder KI: Die Büchse der Pandorra

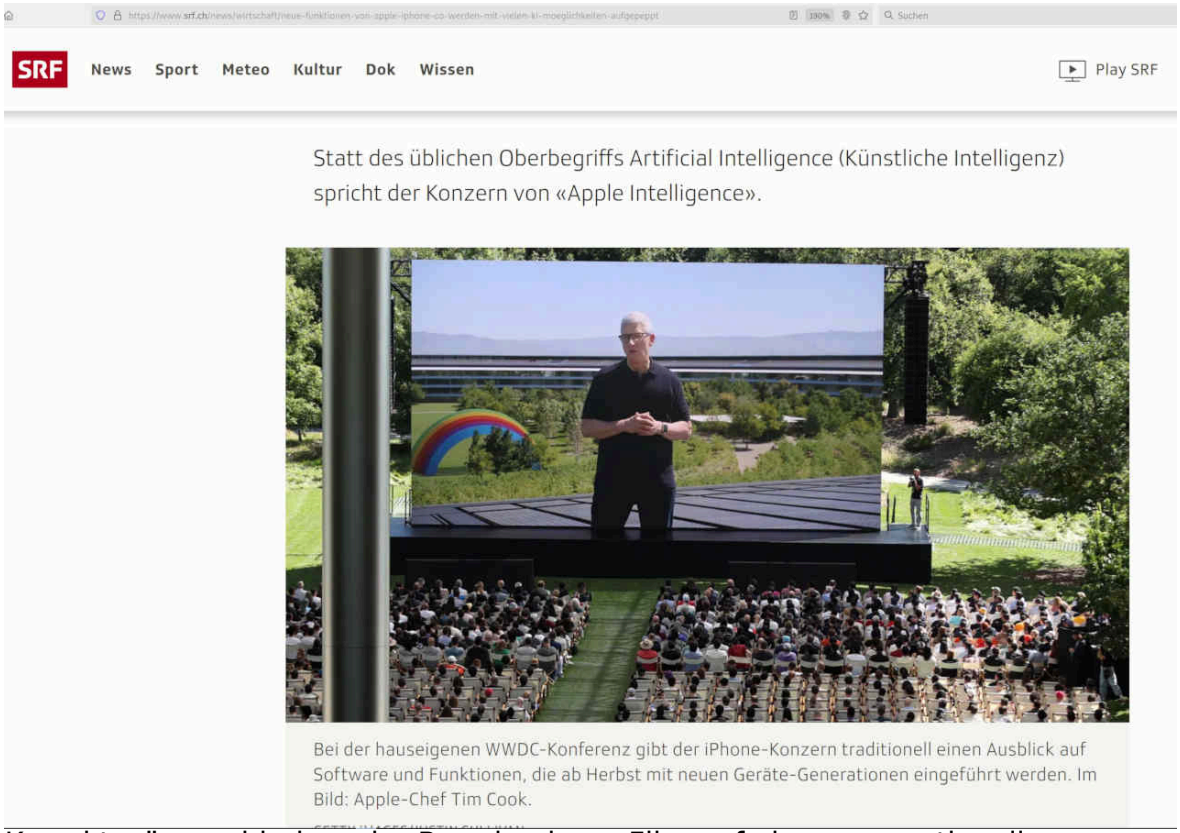
Das hier verwendete **Beispiel der (beinahe) unredigierten Apple-Pressemitteilung, die auf der Hauptseite des Schweizer Fernsehens** landet **(Capt2PDF-Kopie hier),** passt gut zur aktuellen KI-Thematik, weil exemplarisch aufgezeigt werden kann, welche Widersprüche sich aktuell ergeben. Dass eine Privatperson einen solchen Artikel speichern darf, dürfte klar sein. Aber darf ein Betriebssystem die so erstellten Informationen bzw. die daraus gewonnen Informationen in der Cloud speichern?

Wo wäre die KI, wenn nicht milliardenfach urheberrechtlich geschützte Inhalte für das Trainieren der KI verwendet würde? Wie können sich Urheber/innen dagegen wehren, dass ihre Inhalte durch Dritte verwendet werden? Beim hier vorgestellten Tool Capt2PDF erfolgen alle Arbeiten auf einem lokalen Computer. Was passiert, wenn diese Daten den lokaen Computer verlassen?

Enthält z.B. die hier mit **Capt2PDF erstellte Kopie** urheberrechtlich geschützte Inhalte? Was passiert, wenn ein Betriebssystem ungefragt derartige Kopien erstellt und diese öffentlich zugänglich werden? Passend zum Thema Urheberrecht sei auf **anwalt.de** bzw. das entsprechend dort zitierte Urteil verwiesen (Zitat):

Ein Werk ist generell nur dann urheberrechtlich geschützt, wenn es ein gewisses Maß an Schöpfungshöhe aufweist – es darf sich also nicht um etwas Alltägliches handeln – und eine persönlich geistige Leistung beinhaltet. Pressemitteilungen

gelten diesbezüglich als sogenannte „kleine Münze“, d. h. als weniger anspruchsvoller Text, der aber gerade noch die unterste Grenze des Urheberrechtsschutzes erreicht (vgl. auch LG Hamburg, Urteil v. 31.01.2007, Az.: 308 O 793/06).
Beim hier **publizierten (Auszug) des SRF-Artikels** erscheint die Sachlage in diesem Kontext klar. Selbst wenn er die urheberrechtliche Schwelle überschreitet, es geht hier um eine Kritik betr. dem Umgang der Medien mit KI bzw. um eine Lösung mit Capt2PDF, um überhaupt aufzeigen zu können, was falsch läuft bzw. wie es besser gemacht werden kann.
Die Büchse der Pandorra besteht darin, dass z.B. das Aufzeichnen bzw. das Erstellen von durchsuchbaren Inhalten aktuell für Nutzer/innen nicht trivial ist. Die Tech-Konerne werden behaupten, genau darum gibt es die KI. Die öffentliche Wahrnehmung (gerade auch in den Medien) ist die, ohne KI geht es gar nicht.
Darum sollte bei solchen «Artikeln» genau(er) hingesehen werden. Wenn Beiträge nicht mit vollem Namen gekennzeichnet sind, bei Formulierungen wie ‚Siri wird noch schlauer‘, es könnte fast der Verdacht aufkommen, die KI wäre hier am Werk gewesen. Dazu passt, dass die Bilder von GettyImages stammen. Bei der ersten Abbildung wird der Text ‚Siri‘ mit ‚AI‘ gemixt (qualitativ definitiv auf KI-Niveau) und bei der zweiten Aufnahme steht: «Im Bild: Apple-Chef Tim Cook«.



Korrekt wäre wohl, dass der Boss in einem Film auf einem sensationell grossen Display erscheint. Selbst wenn der Beitrag urheberische Qualitäten hätte, eine (fast) komplette Publikation erscheint hier angebracht, weil damit aufgezeigt werden kann, wie unscharf Informationen im Zusammenhang mit KI gehandhabt werden. Ohne Tools wie Capt2PDF ist es aber recht aufwändig, dies aufzuzeigen. Nur, wie liegt der Fall, wenn Hersteller von Betriebssytemen mehr oder minder ungefragt die Daten der Nutzer/innen milliardenfach abkupfern? Dabei darf **(passend zur hier publizierten PDF-Datei)** die Frage gestellt werden, was Wahrheit ist? Was, wenn diese Inhalte nicht ganz so sind, wie sie (in der Werbung nicht unüblich) dargestellt werden? Oder was passiert, wenn die Daten

nachträglich verändert werden?
Beim Apple-Chef findet sich ein amüsanter Regenbogen mit einer gewissen Aussage. Was, wenn die KI in gewissen Staaten etwas gegen Regenbogen hat? Wird die Cloud-KI mir in diesen Staaten einem «neutralen» Wasserstrahl präsentieren? Oder werden unliebsame Informationen gleich in der Echtzeit bei der Anzeige gefiltert?
Welche Chancen haben normale Anwender/innen heute noch? Gegenüber den Tech-Konzernen, die ins Innerste des täglichen Lebens eindringen, gegenüber Medien, welche vielleicht auch etwas kritischer berichten dürften? Gegenüber staatlichen Behörden, bei denen es sehr lange geht, bis datenschutz- bzw. wettbewerbsrechtliche Missstände geahndet werden, wenn überhaupt.
Open Source, so sie denn lokal zum Einsatz kommt, stellt hier einge gute, die wohl beste, Alternative dar. Die Linux-Distribution AVMultimedia ist genau dafür gedacht, Capt2PDF steht hier ab Version 2024/VI so zur Verfügung, das alleine die Nutzer/innen entscheiden, ob Capt2PDF zum Einsatz kommt oder nicht. Kurz und gut, ein zweckmassiges Programm, um Bildschirminhalte so aufzuzeichnen, dass die Büchse der Pandorra nicht geöffnet werden muss. Will heissen, ganz ohne KI und Cloud, dafür einfach, transparent (100% Open Source) und lokal und in Echtzeit auf dem eigenen Linux-Desktop.