Brand-New-Life
Magazin für Kunstkritik

www.brand-new-life.org
office@brand-new-life.org

# Reading Between the Lines

## A Speculation on Material Implications of Hovering Thumbnails on YouTube

Yann Martins

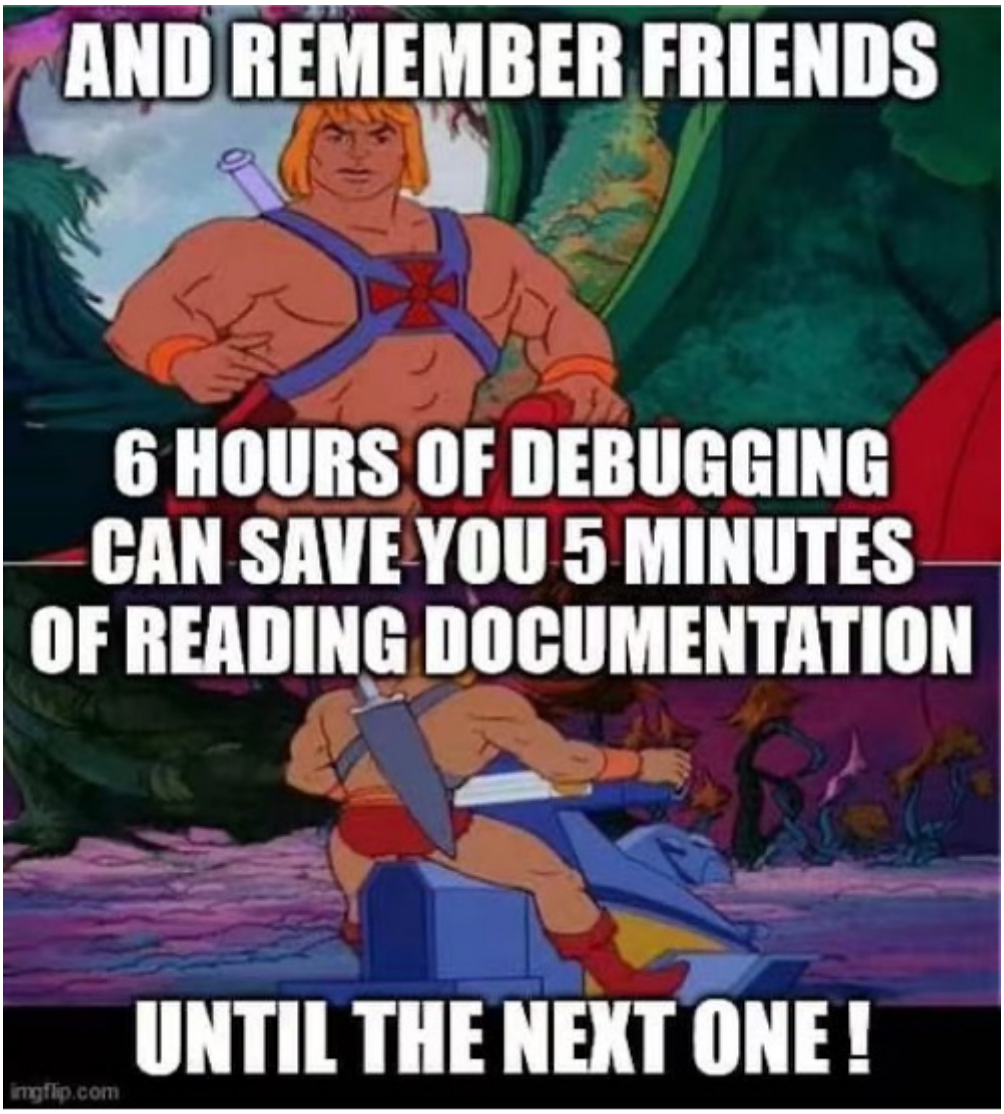
This text is a recount of exploring the algorithmic intricacies of hovering interactions on YouTube video thumbnails. The explorative debugging practice used offers a glimpse into the material implications of hovering, seeing it as a form of exploited labor. Each hover activates a convoluted set of instructions resulting in hardcoded values for what is considered productive and unproductive hovering.

Critical Debugging is a critical aspect of my practice and research. I investigate the mechanisms behind data manufacture on social media platforms, setting YouTube as my case study. By examining the processes through which data is produced within such digital environments, the research aims at shedding light on the intricate dynamics of social media platforms as sites of data production. This text offers a speculation on the material implication such algorithmic processes might have.

Several scholars refer to the production of data through interactions with social media and other online interfaces as modes of extraction.[1] Hence, laptop, smartphone and tablets are the sites for data production. Yet finding the exact places—or lines of code—where data is extracted requires a careful reading of technical, and critical, papers describing how YouTube's recommender system works.[2] This helps us understand where to look to identify such places when visiting https://youtube.com. [https://www.youtube.com/]

In my practice, the primary objective was to develop an understanding of how data is manufactured at an algorithmic level. Achieving this required repurposing **debugging** tools in unconventional ways. These tools are available on any browser and provide an extensive suite of capabilities, including network analysis and code inspection. Debugging tools can halt code execution, reveal variable contents, and, crucially, display the entirety of the code running on the device in use. These tools are commonly employed by web developers for error correction in the process of developing a website. This functionality makes them instrumental for examining the underlying mechanisms of data manufacture.

Drawing on insights from the technical and critical papers cited above, it became feasible to navigate and interpret YouTube's codebase effectively. By employing a combination of methodologies, including network analysis, stack tracing, and logging, a focused investigation over several months led to the identification of one data production site within YouTube's codebase: Hovering




## Annotating Hovering[3]

```
/**
 * This annotated code demonstrates how
 * hover interactions are captured.
 * It specifically tracks when a user
 * hovers over a video thumbnail,
 * triggering a preview of the video content.
 */

e.hideThumbnail = function () {
  ////////////////////////////////////////////////////
  ////////////////////////////////////////////////////
  this.data &&
  this.data.enableHoveredLogging &&
  this.logEvent('INTERACTION_LOGGING_GESTURE_TYPE_HOVER');
  // ??????????????????????????????
  // here the data production starts
  this.removeVideoPreview()
  ////////////////////////////////////////////////////
  ////////////////////////////////////////////////////
};
```

The code above shows the capturing of hover data in https://www.youtube.com [https://www.youtube.com/s/desktop/1857023c/jsbin/desktop_polymer_inlined_html_polymer_flags.vflset/desktop_polymer_inlined_

The annotated code snippet above combines machine-readable algorithms with human-readable explanations. Human-readable text is often marked with double slashes (//) for shorter comments or enclosed within /***/ for more detailed notes. You can see the latter

style used in the opening lines of the snippet for longer explanations.

This explanation serves both as a guide to understanding the code and an invitation to explore it more deeply. By revisiting the snippet, you can focus on the annotations and engage with them as an integral part of the text, enhancing your understanding of the code's structure and purpose.

```
/**
  * line 107813
  * this is the logging function
  * it computes the hovering time
  * and then sends this information
  * back to YouTube servers as data
  */

e.logEvent = function (a) {
  if (!(0 >= this.loadingStartTimeMs)) {
    var b = {
      isMovingThumbnail: this.hasVideoPreview
    };

    ////////////////////////////////////////////////
    // from here the code becomes a bit more readable
    // and it hints at the calculation of how much time
    // the user spends hovering over a thumbnail

    if (this.hasVideoPreview && 0 < this.startTimeMs) {
      var c = this.loadingEndTimeMs - this.loadingStartTimeMs;
      0 < c && (b.movingThumbnailLoadingDurationMs = Math.round(c));

      /*************************
      * calculating hovering time
      *************************/

      b.durationHoveredMs = Math.round(Hj() - this.startTimeMs)

      // ??????????????????????????????????????????
      // here the amount of time of hovering is computed
      ////////////////////////////////////////////////
      ////////////////////////////////////////////////
    }
    this.videoId && (b.videoId = this.videoId);
    this.csn &&
    this.trackingParams &&
    GRa(this.csn, mo(this.trackingParams), a,
    {
      thumbnailHoveredData: b
      // ??????????????????????????????????????????
      //here the measurement of hovering time is saved
    });
    this.startTimeMs =
    this.loadingEndTimeMs =
    this.loadingStartTimeMs = - 1;
    this.hasVideoPreview = !1
  }
};
```

The two code snippets above are just a minuscule part of the extensive codebase that YouTube runs on users' browsers. The first part shows how a log event is triggered: this.logEvent('INTERACTION_LOGGING_GESTURE_TYPE_HOVER').
The second part shows what happens within the logEvent function. The latter computes the amount of time that the user has been hovering over a thumbnail. It saves that data together with the video ID and then sends it over the network back to YouTube's servers.

The code above describes how an action like hovering could be captured and rendered as datapoints. However, for it to be recognized as form of labour, and thereby justify terms like data extractivism, something more needs to happen. First there is the need to acknowledge that data is a commodity that is sold, and that its production requires labour.[4] Sociologist Christian Fuchs describes labour within the digital realm in line with Marxist materialism. In

his view, within labour there is an important separation: necessary labour time and surplus labour.[5]

These two terms separate the moment in which a worker exceeds the threshold of working hours. This threshold defines an economic boundary that separates the number of hours a worker needs to work for their own salary, and the number of working hours that become productive for the employer, producing surplus value. That is, of the 8-hour workday in Western countries the first 4 hours of work might cover the salary and infrastructural expenses of an employer, while over the remaining 4 hours workers produce value for the employer.[6]

Zooming out from a regular company and zooming into the algorithm annotated above an obvious question to be asked of the algorithm is: when does the algorithm become productive? That is, how much time should one be hovering to create a surplus for YouTube.

# Hard coding the boundaries of surplus value production

```
f.then(function () {
  var h = Ak() - b,
  // hard coded values within YouTube hovering
  // recognize labour: below are the hardcoded values
  // for when hovering is productive and when it is not
  /**
  * I want to draw attention to two values 500 and 600000.
  * In computer coding jargon, they are called hard coded values.
  * They are immutable.
  * Usually, hard coded values are what programmers try to avoid,
  * except for specific purposes, like the one shown below.
  */
  l = qj('minimum_duration_to_consider_mouseover_as_hover', 500),
  // half second
  m = qj('max_duration_to_consider_mouseover_as_hover', 600000);
  // 10 minutes

  /**
  * This leads to a paradoxical situation where all
  * analysed code variables were obfuscated
  * by using combinations of 2 ~ 3
  * letters (see for example “l = qj(...)”)
  * in place of proper naming conventions.
  * And yet here where more secrecy would be expected
  * there is no use of variables,
  * and the strings are human readable.
  * ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  * And this readability clearly gives away what is happening
  * in a few lines of code.
  * A value related to hovering over a video thumbnail is
  * measured against these hardcoded values.
  * It makes a difference whether the user is hovering for more
  * than 1/2 second [500ms] or less than 10minutes [600000ms].
  * ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  * Those two values represent two hard boundaries,
  * for what is considered a «mouseover» event, as the strings
  * in clear view are telling us.
  * ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  * Given the infrastructural magnitude of YouTube,
  * it is fair to assume that such hard coded values
  * are not casual.
  * And so, those values left so visible for everyone to read must
  * represent the measurements for what can be considered productive,
  * and unproductive, hovering over thumbnails.
  * They define hard thresholds «to_consider_mouseover_as_hover»,
  * describing a timespan of productive data production.
  */

  h = Math.round(h);
  l > h || m <= h ||
  (l = a.getScreenLayer ? a.getScreenLayer():void 0, l=so(l)||'',
    m = aI($H.getInstance(), a),
    BRa(l, a.visualElement ? a.visualElement : mo(m),
      'INTERACTION_LOGGING_GESTURE_TYPE_HOVER', {
      hoverData: {
```

```
        durationHoveredMs: h
      }
    }));
```

Within these two temporal boundaries, hovering can be understood as a specific form of interaction that contributes to the production of surplus value. These boundaries delineate the timeframe during which a user's activity actively supports the platform's profitability. It can be speculated, for instance, that hovering for less than 500 milliseconds may not generate sufficient engagement to offset the costs associated with server infrastructure. Conversely, extended hovering—exceeding 10 minutes—could potentially impact the platform's revenue model to such an extent that it might influence broader organizational outcomes, such as staffing decisions for software developers or even leadership changes at the executive level.

And so, a final question lingers: are those code snippets just algorithms, or do they represent a contract between users and platform owners?

It is through the—aforementioned—(joyfully) catastrophic speculations that this question could be partially answered. Not so much in the sense that those algorithms really represent a contract. But rather this idea of the contract should be understood symbolically as to represent those algorithms describing when gestures on an interface are considered as labour.

```
1416 c.PropTypes = c;
1417 e.exports = c
1418 }), null);
1419 __d('React', [
1420 'React-dev',
1421 'React-prod',
1422 'create-react-class/factory',
1423 'prop-types',
1424 'ReactFbPropTypes'
1425 ], (function (a, b, c, d, e, f) {
1426 __p && __p();
1427 a = b('React-prod');
1428 var g = a.__SECRET_INTERNALS_DO_NOT_USE_OR_YOU_WILL_BE_FIRED.ReactCurrentDispatcher;
1429 function h(a) {
1430 var b = g.current;
1431 return b.readContext(a)
```

Code from FaceBook.com, found with students during a Counter Data Practices class in 2019

[1] The term extraction unfortunately implies the notion that data exists as raw material, that of course is not true. See Jathan Sadowski, «When data is capital: Datafication, accumulation, and extraction.» in: *Big Data & Society*, 6(1), 2019. https://doi.org/10.1177/2053951718820549;
Yet the term offers a good way to analyze the labour relations, with emphasis on the exploitative aspects that extractive practices employ. See Shoshana Zuboff, *The age of surveillance capitalism: The fight for a human future at the new frontier of power* (First edition). New York: PublicAffairs, 2019; Sandro Mezzadra & Brett Neilson «On the multiple

frontiers of extraction: Excavating contemporary capitalism.» in: *Cultural Studies*, 31(2-3), 185–204, 2017. https://doi.org/10.1080/09502386.2017.1303425

[2] Paul Covington, Jay Adams & Emre Sargin, «Deep Neural Networks for YouTube Recommendations.» *i*n: *Proceedings of the 10th ACM Conference on Recommender Systems*, 191–198. Boston, Massachusetts, USA: ACM, 2016. https://doi.org/10.1145/2959100.2959190; Karin van Es, «YouTube's Operational Logic: ‹The View› as Pervasive Category.» in: *Television & New Media*, 21(3), 223–239, 2020. https://doi.org/10.1177/1527476418818986; Sophie Bishop, «Anxiety, panic and self-optimization: Inequalities and the YouTube algorithm.» in: *Convergence*, 24(1), 69–84, 2018. https://doi.org/10.1177/1354856517736978

[3] What follows is a refined and polished re-enactment of my code annotation process during my debugging practice. These annotations were instrumental in helping me trace the exact steps the code takes to capture user's hovering interactions.

[4] Jathan Sadowski, «When data is capital: Datafication, accumulation, and extraction.» in: *Big Data & Society*, 6(1), 2019. https://doi.org/10.1177/2053951718820549; Shoshana Zuboff, *The age of surveillance capitalism: The fight for a human future at the new frontier of power* (First edition). New York: PublicAffairs, 2019; Sandro Mezzadra & Brett Neilson «On the multiple frontiers of extraction: Excavating contemporary capitalism.» in: *Cultural Studies*, 31(2-3), 185–204, 2017. https://doi.org/10.1080/09502386.2017.1303425

[5] Christian Fuchs, *Digital Labour and Karl Marx*. New York: Routledge, 2013. https://doi.org/10.4324/9781315880075

[6] ibid.

YANN MARTINS

Yann Martins is a debugger and researcher currently working at IXDM. His software practices lie at the intersection of debugging, computer music and game design. He is currently working on his doctoral thesis, in which he investigates debugging practices and browser performances as forms of data poisoning.



doi.org/10.5281/zenodo.15386632