Task 1

1. If file has been uploaded – notify the user right away, do not make him wait till the whole process finishes, especially, if it's not really required from him to get the final result right away.
2. If User does need to get result of the file processing – update the status online: use a static webpage referencing his particular file (in case window gets closed) and update the status online while on that page (SSE can be used here, which should be triggered only if processing has not reached its final stage).
3. Do not store the file in database. In majority of cases, this does degrade performance and is totally unnecessary. Store file on file system with reference to the file in DB (path to file is usually enough). If you do want to store it in DB – store the "decoded" version of the file, that you "split" into columns relevant to your case (unless you need to store the whole document in DB, but then better use MongoDB or the like).
4. Am not sure you need to a neural network to calculate keyword density. Regular preg_match_all will, most likely, serve you just as well and fractions of the cost. Even if you will be running it for each keyword in a loop. Unless the "neural network" is doing just that and not a FULLTEXT-like search.
5. Do not generate the report automatically. Store relevant data in the database and then use it to generate report on-demand. If notification is required – send it, but with a link to webpage, that will generate the report (and preferably link it to the submitter somehow, if that is relevant).

Task 2

1. Limit avatars size (not dimensions, although that can contribute).
2. Use pagination if the data shown can be split without losing its value. Most likely 50 items per page will be more than enough, but it depends on the data and purposes.
3. If data is not changed frequently, use server-side caching. Consider using proper Cache-Control headers as well to utilize client-side caching as well.
4. Check the query used to grab the data. And the table itself: perhaps you are missing indexes or are not using it or use overcomplicated unoptimized JOINs.
5. Utilize lazy loading, if possible and above steps do not improve the situation to appropriate extent. Note, that lazy loading can increase number of roundabout trips to server, though while utilizing separate connections for each. As alternative you can try using HTTP2 push functionality with links to images being sent in HTTP headers (can use https://github.com/Simbiat/HTTP20).

Task 3

I would need these:

1. My https://github.com/Simbiat/Cron (or something similar)
2. A mailer (like https://github.com/PHPMailer/PHPMailer)
3. A \`task\` entry in Cron table linked to mail sending with arguments taken from scheduled version of this task (as per Simbiat/Cron specification)
4. A class (or a set of classes) to handle "business logic"
5. Actual Cron (or another task scheduler)

Once the class needs to send a notification it checks first, whether user has not unsubscribed from these notifications, if no – schedule appropriate task using Simbiat/Cron with high enough priority. If it's one time – schedule with \`schedule\` set to 0, otherwise – with appropriate number of seconds (and other settings). Then the Cron should trigger appropriate tasks to send mail notifications. If user does something that needs to suppress these recurring notifications, the class needs to simply remove the task from schedule.

In case you think, this does not work – it does work on https://www.simbiat.ru/fftracker, where every visit of “outdated” pages creates a task to update the entity there, which is then handled on backend separately. Although they are one-time. But there are other tasks, that are schedule (for example to update statistics, which is done daily). No mail notifications for now, though.